Development of deep learning-based tools for the design of new compounds with desired biological activities

Tiago Sousa

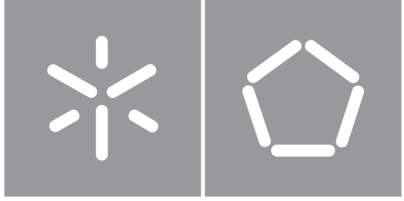
UMinho | 2021

Universidade do Minho
Escola de Engenharia

Tiago Filipe Escairo Sousa

Development of deep learning-based tools for the design of new compounds with desired biological activities

abril de 2021



Universidade do Minho
Escola de Engenharia

Tiago Filipe Escairo Sousa

Development of deep learning-based tools for the design of new compounds with desired biological activities

Dissertação de Mestrado
Mestrado em Bioinformática
Ramo de Tecnologias de Informação

Trabalho efetuado sob a orientação do
Miguel Francisco Almeida Pereira Rocha, Associate Professor, Department of Informatics, University of Minho

Vitor Manuel Sá Pereira, Post-doc Researcher, Centre of Biological Engineering

abril de 2021

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

AGRADECIMENTOS

Gostaria de agradecer ao meu orientador, o Professor Miguel Rocha, e ao meu co-orientador, o Dr. Vítor Pereira, por toda a ajuda e apoio que me deram e por todo o conhecimento que transmitiram. Espero que a qualidade desta dissertação reflita os ótimos padrões que sempre me incentivaram a atingir.

Ao João, Ruben e restante BiSBII, agradeço pela ajuda nas mais variadas questões, pelo vosso trabalho que contribuiu para os resultados aqui apresentados, e por todas as discussões interessantes que fomos tendo.

Aos meus pais agradeço o apoio e carinho incondicional. Os vossos esforços para me dar as melhores oportunidades e condições são a razão de poder completar esta etapa e a vossa educação, a razão de ser quem sou. Muito obrigado por tudo.

À minha restante família, a minha madrinha, padrinho, primos, tias, tios, avôs e avó, agradeço-vos pelos bons momentos, companhia e suporte.

À Patrícia, agradeço por adoçares a minha vida com a tua companhia e por trazeres uns raios de luz para os momentos mais amargos. Obrigado por me acompanhares em todas as minhas aventuras e me guiares quando perco o rumo. Tudo o que passamos juntos é inesquecível.

Aos meus amigos, Ricardo, Beatriz e Joana, que apesar de longe me acompanharam nesta etapa, e à Daniela, Gabriela, Sara e Pedro por partilharmos o que há de bom na vida, o meu grande obrigado.

À Universidade do Minho e seus muitos colaboradores, aos profissionais de saúde, de segurança e de manutenção, e à sociedade no geral, agradeço por proporcionarem uma estrutura social funcional que permite o desenvolvimento de investigação científica livre e democrática. Gostaria de agradecer especialmente ao programa de investigação Horizonte2020-Shikifactory100, por financiar a investigação aqui desenvolvida, através da bolsa nº 814408, e aos investigadores com os quais tive o prazer de colaborar neste contexto e que contribuíram para os resultados aqui apresentados.

ABSTRACT

In the last few years, *de novo* molecular design has increasingly been using generative models, from the emergent field of Deep Learning (DL), to propose novel compounds that are likely to possess desired properties/activities, in areas such as drug discovery, materials sciences or biotechnology. A panoply of deep generative models, such as Recurrent Neural Networks, Variational Autoencoders, Adversarial Autoencoders and Generative Adversarial Networks, can be trained on existing datasets, and provide for the generation of novel compounds, typically with similar properties of interest. Additionally, different optimization strategies, including transfer learning, Bayesian optimization, reinforcement learning, and conditional generation, can be used to direct the generation process towards desired aims, regarding their biological activities, synthesis processes or chemical features. Various instances of experimental validation of these emerging methods have surfaced, with *de novo* generated molecules being synthesized and proving successful in *in vitro*, and even *in vivo*, assays. These successful practical realizations encourage further research into this blooming field.

This dissertation aims to explore the application of generative DL to the *de novo* molecular design, with a focus on the targeted generation of new compounds. Two frameworks were developed to support this endeavor and stand as the main contributions of this work. The first, termed DeepMolGen, standardizes the implementation and usage of various generative DL architectures for molecular design. The second, termed EAMO, employs multi-objective evolutionary algorithms to navigate the latent space of autoencoder based models, optimizing the generation of molecules with desired characteristics. These frameworks were accompanied with a systematic and critical review on deep generative models, the related optimization methods for targeted compound design, and their applications.

Four state-of-the-art architectures were implemented, trained and evaluated under the DeepMolGen framework using a standard dataset and common metrics such as validity, uniqueness, novelty and the MOSES benchmark. The results showed that DeepMolGen was capable of performing the intended tasks and that most of the implemented models performed on par with their publications. Similarly, four case studies from the literature were optimized with EAMO and the results compared to previous works. These experiments showed that EAMO could control abstract chemical properties and is competitive with other state-of-the-art methods. Lastly, the three best performing models were combined with transfer learning and EAMO within a pipeline for the generation of sweeteners. The resulting set of 102 promising molecules was reviewed by expert chemists and the pipeline improved with their feedback. A second set of 99 compounds was then generated and the preliminary observations pointed to significantly improved results.

Keywords: Deep Learning · Generative Models · Molecular Design · Multi-objective Evolutionary Algorithms · Novel Sweeteners

RESUMO

Ao longo dos últimos anos, a criação de moléculas *de novo* tem vindo cada vez mais a utilizar modelos generativos, da área do *Deep Learning* (DL), para propor compostos com propriedades/atividades de interesse em áreas como descoberta de fármacos, ciências dos materiais ou biotecnologia. Uma panóplia de modelos DL, que incluem arquiteturas como *Recurrent Neural Networks*, *Variational Autoencoders*, *Adversarial Autoencoders* e *Generative Adversarial Networks*, podem ser treinados com conjuntos de dados existentes permitindo a geração de novos compostos, tipicamente com propriedades de interesse semelhantes. Adicionalmente, várias estratégias de otimização, incluindo *transfer learning*, otimização Bayesiana, aprendizagem por reforço e geração condicionada, podem ser utilizadas para guiar o processo de geração em direção a propriedades de interesse como atividade biológica, processo de síntese ou características químicas. Têm surgido ainda vários exemplos de validação experimental destes métodos, nos quais moléculas geradas *de novo* são sintetizadas e demonstram sucesso em ensaios *in vitro* e *in vivo*. Estes sucessos práticos encorajam investigações adicionais nesta área emergente.

A presente dissertação pretende explorar a aplicação de DL generativo para o desenho de moléculas *de novo*, com um foco na geração direcionada de novos compostos. Duas *frameworks* foram desenvolvidas para este propósito e constituem as principais contribuições deste trabalho. A primeira, DeepMolGen, padroniza a implementação e utilização de variadas arquiteturas de DL para o desenho molecular. A segunda, EAMO, aplica algoritmos evolucionários para navegar o espaço latente de modelos baseados em *autoencoders*, otimizando a geração de moléculas com características pretendidas. Estas *frameworks* foram acompanhadas de uma revisão sistemática sobre modelos generativos de DL, métodos de otimização para a geração direcionada de compostos, e as suas respetivas aplicações.

Quatro arquiteturas do estado-da-arte foram implementadas, treinadas e avaliadas com o DeepMolGen, usando um conjunto de dados standard e métricas comuns como validade, unicidade, novidade e o conjunto de testes MOSES. Os resultados mostraram que o DeepMolGen conseguiu realizar as tarefas pretendidas e que a maioria dos modelos comportaram-se de forma semelhante às respetivas publicações. De forma semelhante, quatro casos de estudo da literatura foram otimizados com o EAMO e os resulta comparados com publicações prévias. Estas experiências mostraram que o EAMO é capaz de controlar propriedades químicas abstratas e que é competitivo com outras abordagens do estado-da-arte. Por fim, os três melhores modelos foram combinados com *transfer learning* e o EAMO para abordar a geração de compostos adoçantes. O conjunto de 102 moléculas resultante foi revisto por especialistas em química e a metodologia melhorada com os comentários. Um segundo conjunto de 99 compostos foi então gerado e os comentários preliminares apontaram para uma melhoria significativa dos resultados.

Palavras Chave: Algoritmos Evolucionários Multi-Objectivo · Deep Learning · Desenho Molecular · Modelos Generativos · Novos Adoçantes

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AAE Adversarial Autoencoder. 2, 21, 22, 32, 36, 37, 41, 48, 61–63, 65, 66, 77, 87

AE Autoencoder. 1, 2, 21, 22, 32, 36–40, 48, 51, 52, 54, 61, 63, 64, 67, 68, 71, 78, 87, 88

AI Artificial Intelligence. 2, 11

ANN Artificial Neural Network. 2, 15–17, 21

BO Bayesian Optimization. 2, 36, 37, 45, 89

CLaSS Conditional Latent Attribute Space Sampling. 2, 38, 44

CNN Convolutional Neural Network. 2, 17, 18

DCN Densely Connected Network. 2, 17, 18

DL Deep Learning. v, vi, 1, 2, 27, 32, 35, 43, 45–47, 59, 87, 88

DNN Deep Neural Network. 2, 17, 56, 70, 76

DRD2 Dopamine Receptor D2. 2, 33–35, 37, 38, 42, 56, 58, 69, 70, 74

EA Evolutionary Algorithm. 2, 26, 27, 54, 56–59, 74, 75, 80, 82, 84, 85, 87–89

ECFP Extended Connectivity FingerPrints. 2, 7

FCD Fréchet ChemNet Distance. 2, 29–31, 68

GA Genetic Algorithm. 2, 37, 55

GAN Generative Adversarial Network. 1, 2, 20, 22, 27, 28, 32, 35, 36, 38, 41

GDE3 Generalized Differential Evolution 3. 2, 55, 74, 75

GGNN Gated Graph Neural Network. 2, 28

GNN Graph Neural Network. 2, 28, 32, 39

GRU Gated Recurrent Units. 2, 19, 20, 61, 62

InChI International Chemical Identifier. 2, 6, 7, 27

IUPAC International Union of Pure and Applied Chemistry. 2, 5, 7

logP logarithm of the octanol–water partition coefficient. 2, 5, 35, 36, 41, 43, 44, 55, 69, 70, 84

LSTM Long Short-Term Memory. 1, 2, 19, 20, 27, 33, 62–64

ML Machine Learning. 2, 11, 12, 54, 56, 76

MLP Multi Layer Perceptron. 2, 16, 28

MO Multi-Objective. 2, 38, 55, 57, 69–75, 78, 84, 88

MSE Mean Squared Error. 2, 14

NSGA-II Non-dominated Sorting Genetic Algorithm II. 2

NSGA-III Non-dominated Sorting Genetic Algorithm III. 2, 55, 57, 71, 74, 78

PCE Power Conversion Efficiency. 2, 45

PlogP logP penalized by SA and large rings. 2, 34, 36–38, 42, 43, 58, 69–71

PSO Particle Swarm Optimization. 2, 38

QED Quantitative Estimate of Drug-likeness. 2, 34, 36, 38, 39, 41–44, 55, 58, 69–71, 74

RF Random Forest. 2, 15, 56, 76

RL Reinforcement Learning. 2, 11, 34–36, 42, 89

RNN Recurrent Neural Network. 1, 2, 18, 19, 23, 27, 28, 32–34, 39, 43, 48, 61–64

SA Synthetic Accessibility. 2, 36, 39, 42–44, 55, 81

SELFIES SELF-referencIng Embedded Strings. 2, 27, 32, 44, 47, 89

SMILES Simplified Molecular Input Line Entry System. 2, 5, 6, 27–30, 32–39, 41, 43, 44, 47, 48, 51, 62, 89

SO Single-Objective. 2, 55, 57, 69, 71–75

SOM Self-Organizing Maps. 2, 35, 44

SPEA2 Strength Pareto Evolutionary Algorithm. 2, 55, 74

SVM Support Vector Machine. 1, 2, 13, 14, 56, 70, 76

TPSA Topological Polar Surface Area. 2, 39, 41, 69

VAE Variational Autoencoder. 2, 21, 22, 27–30, 32, 35, 36, 38, 39, 41, 44, 48, 61–65, 68, 87

WAE Wasserstein Autoencoder. 2, 42

1

INTRODUCTION

1.1 Context and Motivation

The development of new compounds with desired biological activities, including drugs, food ingredients or other molecules of interest is an expensive process. In average, it costs roughly 2.8 billion dollars to bring a new medicine to market [1]. A major challenge in this process is the so-called *de novo* compound design, the process of finding new molecules with the required physicochemical and/or pharmacological properties. Simply selecting and testing new molecules at random is not possible due to the immense search space of around 10^{33} - 10^{80} feasible molecules [2].

Current approaches either rely on exhaustively testing meticulously assembled libraries of "promising" compounds, on combining predefined groups of atoms or fragments through a set of expert coded virtual reactions, or in optimizing known molecules, using, for example, genetic algorithms [3, 4, 5]. Although these methods have proven effective in many cases, they limit the available chemical space from which molecules can be drawn, missing possible breakthroughs.

Over the last decade, DL has proven successful in many tasks such as computer vision, speech recognition and natural language processing [6]. In recent years, the application of DL for *de novo* compound design has also flourished. Segler et al. [7] applied a transfer learning approach to Recurrent Neural Networks (RNNs), specifically Long Short-Term Memory (LSTM) layers, successfully generating relevant molecules. Following this method, Merk et al. [8, 9] generated, synthesized and tested several molecules, further validating this approach. Guimarães et al. [10] proposed ORGAN, an approach based on Generative Adversarial Networks (GANs), not only capable of generating valid and diverse molecules, but also of optimising various molecular properties. Blaschke et al. [11] compared four different approaches based on Autoencoders (AEs) and proposed using Bayesian optimization, guided by a Support Vector Machine (SVM) classification model, as a way to search the latent space for new active structures. Winter et al. [12] proposed a method for data augmentation when using AEs by training to reconstruct equivalent representations into the same molecule. Alongside these works, several reviews have also sought to condense the plethora of different approaches, shedding light into and discussing the different architectures, generation of various molecular representations and use of comparative metrics [13, 14, 15, 16, 17].

Despite the rapid evolution this field has experienced, with novel methods appearing at an impressive rate, further developments and improvements are still desirable and required for industry-wide implemen-

tation. As such, exploring different architectures, methods for targeted molecular generation as well as other molecular representations remains an interesting avenue of research.

1.2 Objectives

The main goal of this project is to develop generative DL-based tools for the design of new compounds with desired properties regarding their biological activity. More specifically, the technological objectives are:

- Survey the state of the art within the field of *de novo* compound design, exploring and reviewing available methods and tools with a focus on machine learning and DL methods;
- Implement generative models based in recurrent architectures (e.g. LSTM networks), Variational Autoencoders (VAEs), Adversarial Autoencoders (AAEs) and GANs for the design of new compounds with improved biological capabilities;
- Experiment with applying Evolutionary Algorithms (EAs) to navigate the latent space of chemical AEs for the targeted generation of new compounds;
- Test and evaluate the developed tools with different case studies, with a focus on the design of novel sweeteners.

1.3 Dissertation structure

The remaining dissertation is organized in six chapters. It starts with two chapters discussing the relevant theoretical background. Chapter 2 introduces chemoinformatics, various molecular descriptors and representations and also several relevant compound databases. Chapter 3 introduces the Machine Learning (ML) field, covering several useful concepts and subfields. After exploring some more traditional ML models, the field of DL is introduced, with a special focus being placed on the training and regularization of deep networks as well as on the architectures relevant for generative modeling.

A review of the state-of-the-art in *de novo* molecular generation with DL is then presented in Chapter 4. Following a brief discussion of the alternatives to DL, it covers the methods to generate the various molecular representations, the benchmarking of these models, the approaches to targeted generation of molecules and current applications.

Chapter 5 then details the two frameworks developed in this work, one for supporting the implementation of generative DL models, termed DeepMolGen, and the other to target the generation of new molecules towards specific properties, termed EAMO.

Chapter 6 then discusses the application of the aforementioned frameworks with several case studies. It starts by detailing the implementation and evaluation of several state-of-the-art architectures within DeepMolGen. Then, it presents the validation of the targeted generation capabilities of the EAMO framework, comparing it with other state-of-the-art methods. It concludes with the discussion of the design of

novel sweeteners, covering the process to generate a first set, the feedback from expert chemists and the resulting improved pipeline and second set.

Finally, Chapter 7 presents a summary of this work and the resulting contributions, draws the main conclusions and provides an outlook on future directions of research.

2

CHEMOINFORMATICS

"Chemoinformatics is the application of informatics methods to solve chemical problems"[18]

The application of computational methods to chemistry dates back several decades gaining special prominence from the 1960s onwards [19]. More recently, around 20 years ago, chemoinformatics was first introduced as a drug discovery specific field, but it was quickly adopted to reference the more generic and inclusive "application of information technology to chemistry" [20]. In the following years, the rapid increase in available chemical information and computational power drove further interest in this field [19]. In the general sense, chemoinformatics deals with the organization and analysis of chemical data, such as creating and searching massive chemical databases or predicting certain properties based on the structure of a compound [20].

In chemoinformatics, describing and analyzing certain properties of molecules and encoding molecules in a suitable form for computation, using molecular descriptors and molecular representations respectively, is of particular interest.

2.1 Molecular Descriptors

Molecular descriptors are numerical values that capture the properties of a molecule. They can reflect physicochemical properties such as hydrophobicity or structural features such as aromatic rings. These can be useful for estimating a molecule's behavior in some given conditions or for aiding in the evaluation of the similarity of two molecules [21].
There are currently thousands of different molecular descriptors available, some of which are described below [22].

2.1.1 *Physicochemical Descriptors*

Physicochemical descriptors detail properties that are experimentally determinable and that sometimes can also be estimated computationally. These include [21]:

- molecular weight;
- molar refractivity;

- melting point;
- logarithm of the octanol–water partition coefficient (logP).

2.1.2 *Structural Descriptors*

Structural descriptors describe topological features readily determinable from the structure of the molecule alone. They are usually divided by their respective dimensionality, and include [18, 22]:

- 0-D - Constitutional (number of atoms, number of bonds);
- 1-D - Structural keys (fragment counts, counts of atom types)
- 2-D - Topological (number of molecular walks, connectivity indices chi, kappa shape indices);
- 3-D - Geometrical (polar surface area, radial distribution function, molecular eccentricity).

2.2 Molecular Representations

Before analyzing and processing chemical data, we first need to store chemical compounds in a form suitable for computer processing. Although trivial names such as "benzene" and "caffeine" are easy to remember, they carry little to no information on the structure and properties of the underlying compound. Systematic nomenclature such as the International Union of Pure and Applied Chemistrys (IUPACs) names can be very lengthy and not specify the full structure. As such, several notation systems have been developed to provide a suitable form of representing molecules [18].

2.2.1 *SMILES*

Simplified Molecular Input Line Entry System (SMILES) is a notation language for representing molecules as sequences of characters. It encodes the atoms in a molecule and how these connect with each other in a simple and human readable form. It was originally developed in 1986 by Weininger et al. [23] and its basic ruleset is:

- Atoms are encoded as their atomic symbol, such as Carbon 'C' or Bromine 'Br';
- Bonds are represented as '-' for single, '=' for double and '#' for triple bonds;
- Branches are enclosed with parenthesis, such as '(CO)';
- Rings are denoted by a number for both their opening and closing, such as 'CO1cccc1CO';
- Atoms inside rings are written in lower case.

As encoding a molecule into a SMILES can start at different locations on the molecule, there is a one-to-many relationship where several different SMILES can represent a single molecule. As such, canonicalization algorithms have been developed to ensure that a one-to-one relationship is possible. Figure 1 shows the canonical SMILES of acetaminophen. Nevertheless, these are implementation dependent and a canonical SMILES obtained from one algorithm may not match that of a different algorithm [18].

CC(=O)Nc1ccc(O)cc1

Figure 1: Canonical SMILES of Acetaminophen. Some noteworthy features are: two branchs '(=O)', '(O)' and a ring '1ccc(O)cc1'.

2.2.2 *InChI*

International Chemical Identifier (InChI) is a notation language that represents molecules as a layered string of characters and aims to be a unique machine-readable representation of a structure [24]. Here, molecules are encoded as predefined layers of information which are arranged in a specific order. InChI starts by specifying a core parent structure to which further information may be added. Each layer is separated by a delimiter '/' and prefixed by a lower case letter identifying the layer (with the exception of the first layer). Here the layers specifying the core structure are:

- Empirical formula layer, with no prefix, encodes the chemical composition;
- Skeletal connections layer, prefixed by '/c', encodes the connections between atoms in the molecule (branches are enclosed in parenthesis);
- Hydrogens layer, prefixed by '/h', lists the bonds between the atoms;

Additional details can be added to the right of the previous layers, using one or a combination of:

- Charge layer, composed of two sub-layers:
  - a charge sub-layer prefixed by '/q' providing information about the net charge of the molecule;
  - a protonation/deprotonation sub-layer prefixed by '/p', informing about alterations to the number of protons and their location in the molecule;
- Stereochemistry layer, encodes the relative spatial arrangement of included atoms. It is divided in two sub-layers:
  - the double bond sub-layer'/b';
  - the tetrahedral sub-layer'/t';
- Isotopic layer, prefixed by '/i', encodes the isotopic changes of the atoms herein specified.

In this way, InChI encodes a molecule as its main structure through the empirical formula, skeletal connections and hydrogens layers and adds explicit features through the remaining layers such as stereochemistry or charge information. An important note is that this set of layers composes the standard InChI as defined by IUPAC in 2009. The fixed H layer, prefixed by '/f' and the reconnected layer, prefixed by '/r', are not included in the standard InChI [24]. Figure 2 shows the standard InChI of acetaminophen.

InChI=1S/C8H9NO2/c1-6(10)9-7-2-4-8(11)5-3-7/h2-5,11H,1H3,(H,9,10)

Figure 2: Standard InChI of Acetaminophen, as denoted by the leading '1s'. Three layer are present: the empirical formula layer '/C8H9NO2', the skeletal connections layer '/c1-6(10)9-7 ...' and the hydrogens layer '/h2-5,11H...'.

2.2.3 *Molecular Fingerprints*

Fingerprints encode a molecule as a bit string. Its construction can follow two main approaches: it can encode molecular properties by incrementally encoding into the bit string value ranges such as the molar weight and solubility, or encode molecular topology with bits representing the presence or absence of particular substructures and features in the molecule [20, 25]. Figure 3 illustrates the later approach.

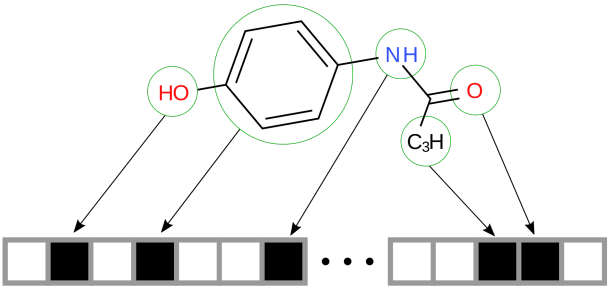


Figure 3: A depiction of a molecular fingerprint where the presence of different structural features is encoded as bits, with a value of 1, in the fingerprint. Bits left turned off, with a value of 0, indicate the absence of features. Although these can be of arbitrary length, 2048 bit fingerprints are common.

Furthermore, the encoding of topological features can be achieved in different ways, depending on how the molecular graph is traversed (examples: path based fingerprints and circular fingerprints). Particularly, in circular fingerprints, circles propagate from each atom for a certain number of bonds, an integer is then computed for each circle and incorporated into the bitstring [25]. Figure 4 portrays this process.

Despite being originally developed as a structural search tool, fingerprints are also widely used to measure similarity between two molecules [20]. Particularly, the usage of Extended Connectivity FingerPrints (ECFP), a type of circular fingerprint focusing on encoding features relevant to molecular activity with similarity metrics such as the Tanimoto coefficient, $T_{(A;B)} = \frac{|A \bigcap B|}{|A \bigcup B|}$, has become a common procedure [25].

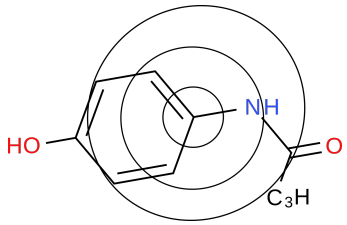
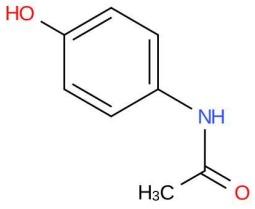


Figure 4: Circular fingerprints on Acetominophen. Circles propagate for 1, 2 and 3 bonds and information is encoded into the fingerprint. A similar process then occurs for each remaining atom in the molecule.

2.2.4 *Molecular Graphs*

A popular solution is to store compounds as graphs, where the nodes represent atoms and edges represent bonds. These molecular graphs are commonly implemented using an adjacency matrix that specifies which atoms are connected and the respective bond order/type, Figure 5. Furthermore, nodes and edges can have properties associated with them, such as relative spatial location and bond order/type, respectively. This format allows the encoding of detailed topological data in a readily processable form [19].



	C	C	O	N	C	C	C	C	O	C	C
C	0	1	0	0	0	0	0	0	0	0	0
C	1	0	2	1	0	0	0	0	0	0	0
O	0	2	0	0	0	0	0	0	0	0	0
N	0	1	0	0	1	0	0	0	0	0	0
C	0	0	0	1	0	1.5	0	0	0	0	1.5
C	0	0	0	0	1.5	0	1.5	0	0	0	0
C	0	0	0	0	0	1.5	0	1.5	0	0	0
C	0	0	0	0	0	0	1.5	0	1	1.5	0
O	0	0	0	0	0	0	0	1	0	0	0
C	0	0	0	0	0	0	0	1.5	0	0	1.5
C	0	0	0	0	1.5	0	0	0	0	1.5	0

Figure 5: Illustration of a molecular graph (left) and its respective adjacency matrix (right). In the last, the absence of a bond is denoted by '0', single bonds by '1', double bonds by '2' and aromatic bonds by '1.5'.

2.2.5 *Three-Dimensional Representations*

Representing a molecule as a simple connection table of atoms overlooks its three-dimensional conformation and, as such, disregards valuable information. Representing the arrangement of atoms in space can be accomplished by coupling a coordinate system to a connection table. A common solution is to represent the molecule inside a Cartesian space, Figure 6 (left), assigning each atom in the connection table three spacial coordinates (x, y, z) [18].

An alternative is to use internal coordinates, Figure 6 (right), such as bond lengths, bond angle, and torsion angle, to describe each atom's position relative to its neighbours, foregoing the need for a fixed coordinate system [18].

Although useful, these two solutions only depict a molecule as a three-dimensional graph without any volume. In reality a molecule has an electron cloud surrounding its atoms from which many of its properties arise.

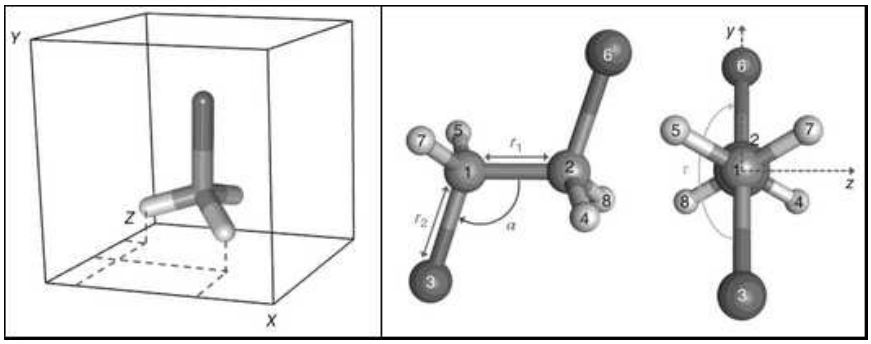


Figure 6: Three-Dimensional representations of molecules. **Right**: coordinates in Cartesian space; **Left**: internal coordinates. Sourced from [18]

In molecular surfaces, a molecule is represented as a closed surface that delimits the volume it occupies. Particularly, this surface outlines a threshold value of electron density in the electron cloud that surrounds the molecule. The molecular surface can also be associated with properties such as the electrostatic or hydrophobicity potential at particular locations [18].

2.3 Databases

There are currently various repositories offering vast collections of molecules in various representations. Some are specialized, offering focused libraries of known active compounds such as drugBank[26], while others store more diverse compounds such as ChEMBL[27]. Several of the commonly used datasets in the de novo drug design were summarized in Table 1.

Table 1: Databases of interest

Database	**Molecules**	**Information**
ChEMBL [27]	2M compounds	Bioactive drug-like small molecules
ExCAPE-DB [28]	1M compounds	Active/ inactive molecules by target
ZINC [29]	750M compounds	Drug-like molecules, available for purchase
PubChem [30]	111M compounds	Mostly small molecules
DrugBank [26]	13K drug entries	Approved and experimental drugs
GDB-17 [31]	166B compounds	Combinatorially generated molecules
REAL database [32]	1.95B compounds	Database of enumerated structures
Tox21 [33]	11K compounds	Toxicity data for various assays
QM8 [34]	22K compounds	Electronic spectra and excited state energy
QM9 [35]	134K compounds	Geometric, energetic, electronic, thermodynamic
PDBbind [36]	17K compounds	3D structures and binding affinity

2.4 Software / Packages

There are currently various software packages and libraries implementing relevant tools frequently used in chemoinformatics. These libraries help with the reproducibility of published work and also ease the development of new approaches. Some examples are:

RDKit [37] - Library in C++ and Python of chemoinformatics and machine-learning tools. The tool is capable of converting between molecular representations, fingerprint calculation, generating various molecular descriptors, etc.

CDK [38] - Library in Java, also available in R, of chemoinformatics and bioinformatics. It is capable of utilizing a variety of molecular representations, 3D geometry generation, substructure search, fingerprint calculation, etc.

Open Babel [39] - Library focusing in supporting and translating between over 110 different molecular representations.

3

MACHINE LEARNING

The field of Artificial Intelligence (AI) seeks to give machines the capacity to think. That is, the ability to autonomously use the right knowledge to further its own goals [40]. ML emerged as a part of AI that deals with self-learning algorithms capable of building knowledge from data. Here, self-learning means that the rules and parameters used for inference are directly derived from the data itself [41]. These algorithms are especially useful when a high volume of data is available [41].

3.1 Types of Machine Learning

The field of machine learning is commonly divided into three subfields, unsupervised learning, supervised learning and reinforcement learning.

Unsupervised Learning deals with unlabeled data and seeks to describe or identify patterns or relationships in it. It is usually employed as an exploratory tool, providing useful insights into the structure of the data without the need for explicit labels. A very common application of unsupervised learning is clustering analysis where one attempts to divide the data into distinct groups or clusters [41, 42].

Supervised Learning deals with labeled data, that is, data where each sample has a corresponding signal or label. It aims to map, or model, the structure of an observation into a label. Depending on this label, this task can be further divided into **regression** (quantitative labels) or **classification** (qualitative labels). The learned models then enables predicting the label of unseen observations. Examples of supervised learning are random forests, neural networks and support vector machines [41, 43].

Reinforcement Learning (RL) outlines a framework where an agent, or system, interacts with an environment through a sequence of actions that are dictated by a policy and evaluated by a reward signal. The agent must then iteratively revise the said policy in order to improve the cumulative rewards over the full sequence of actions. This framework aims at learning a system capable of adopting the best set of actions in a given environment [41, 42].

Generative Learning aims at capturing the underlying data generation process, some unknown probabilistic distribution from which a dataset was sampled. It usually deals with unlabeled, data and attempts to create a model capable of generating new observations that closely resemble those from the training data, but that does not simply copy them. These models must be stochastic in nature, otherwise they would always output the same result and be unable to approximate the intended data generation process [6].

3.2 Basic Concepts and Terms

Before dwelling into more advanced concepts in ML, it is important to define and clarify some concepts and their particular meaning in this context.

Data in ML usually consists of independent observations, or samples, that capture an underlying data generation process. Each of these samples is usually described by a number of features or characteristics [41]. An example of a dataset could be that of a set of known binders and decoys where each sample is a molecule, described by its molar weight and logP or any other set of possible descriptors.

Learning, in this context, pertains to the process of fitting a set of tunable parameters of a model using data to inform the process. This procedure is usually termed model training and follows an optimization framework whose objective depends on the task being performed [42]. Following the previous example, we could attempt to find a set of parameters for logistic regression that best predict active compounds. The logistic regression equation:

$$P(y|x) = \frac{1}{1 + e^{(-b+\sum \theta x)}}$$

represents the probability of a sample x belonging to a class y. Here, b is the bias term, θ is the set of tunable parameters, and x is the vector of the given sample's features.

Hyper-parameters refers to a set of parameters whose values are not directly learned from data. These parameters usually express different model properties and are chosen before the training of the model begins [41]. An example of a hyper-parameter in the case of logistic regression is the choice of solver.

Variance measures the dependency of a model on the particular dataset used during training. In other words, it measures how much the model would vary if it was trained with different data. Ideally we seek low variance models, ones that generalise well from the training data [43]. When this does not happen, and the model has a high variance, it is said to be **overfitting** [41].

Bias measures the error in the predictions of the model, regardless of the dataset used, that is, it measures the approximation error of the function used. Ideally we seek low bias models, ones that are

a good fit to the data [43]. When this does not happen and the model has a high bias it is said to be **underfitting** [41].

Capacity refers to the hypotheses space that a model can cover i.e., its complexity. As a model increases in capacity its bias decreases and it can better approximate the data. Meanwhile, its variance increases and the model depends more and more on the particular training set. Generally, these don't change at the same rate, with initial increases in capacity greatly reducing bias but not dramatically increasing variance. Similarly, after a certain point, further increases in capacity don't significantly reduce bias but greatly increase variance. The turning point represents the desired model, where it has both low bias and low variance, that is, a good **bias-variance trade-off** [43]. Figure 7 depicts this process.

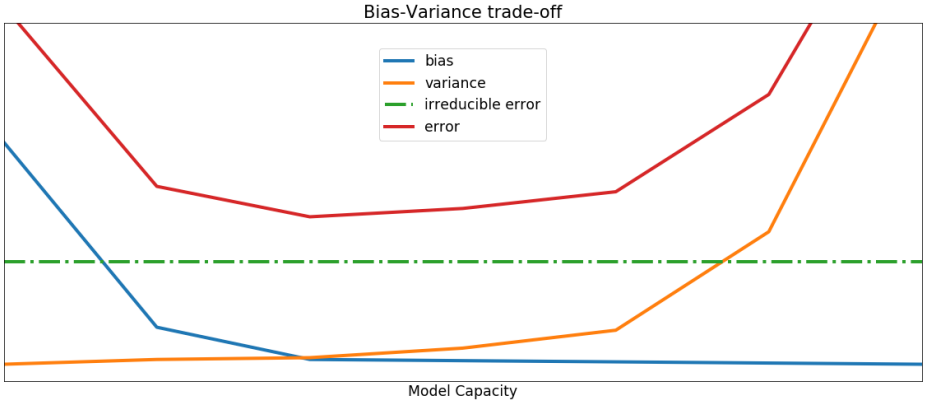


Figure 7: Illustration of the bias variance trade-off. As model capacity increases, bias (blue) decreases and variance (orange) increases, however these don't change at the same rate, allowing for a balance to be found with a low overall error (red). The green dotted line represents the irreducible error in the data, the lower bound of the models error. It can be due to unmeasured, or even unmeasurable, variables.

3.3 Supervised Machine Learning

3.3.1 *Support Vector Machines*

Support Vector Machines (SVMs) are models which try to find a decision boundary that can accurately separate the samples while maintaining the widest margin or distance between the boundary and its nearest samples [43].

In practice, a perfect separation may not be possible and, as such, this problem may not be solvable. To overcome this issue, SVMs use soft margins which allow for samples to be inside the margin or even be misclassified, penalising the solution accordingly [42, 43]. The task is [42]:

$$min \ \frac{1}{2}\|w\|^2 + C\sum_t \xi^t$$

$$subject\ to: \ r^t(w^t x^t + w_0) \geq 1 - \xi^t \ , \ \forall t$$

where, on the left hand term of the equation, w represents the width of the margins and, on the right hand term, ξ represents the soft margins. The optimization is subject to constraints ensuring that the hyperplane allows classifying the samples correctly [42]. The C hyperparameter controls the soft margin width and the number of samples that can be misclassified. Generally, larger C values lead to models with a lower bias but higher variance, while smaller C leads to models with lower variance but higher bias. This hyperparameter controls the bias-variance trade-off being usually chosen using cross-validation [43].

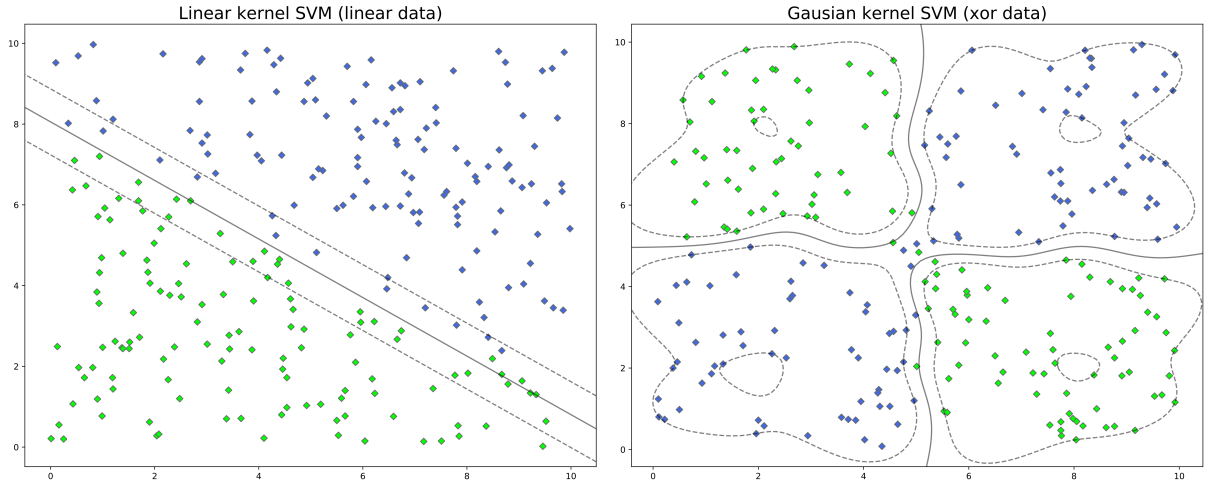


Figure 8: Decision boundary and support vectors found using a linear kernel on linear data (left), and a gausian kernel on XOR data (right).

Although SVMs find a linear decision boundary they can adapt to non linear relationships in the data through the use of kernels, which expand the feature space with non-linear combinations of the original features. Although the hyperplane in this new, higher dimensional, space remains linear, it becomes non-linear in the original space. The key advantage of SVMs is that this procedure can be done in a tractable fashion thanks to the application of kernel transformations (whose definition is beyond this introduction). There are a number of kernels available, a popular one is the radial basis function kernel, or gausian kernel [42]. The left side of Figure 8 illustrates the application of this kernel.

3.3.2 *Decision Trees and Random Forests*

Decision trees are supervised models that recursively separate data through a series of rules or decisions each aiming to maximize the information gain [41]. As Figure 9 shows, a tree is comprised of decision nodes that split data and terminal leaves that label data [42]. It is constructed using a top-down greedy approach. Top-down because it starts with all observations at the tree's root node, progressing downwards as it adds new rules, and greedy because each decision is chosen based on an immediate reward of how well it separates samples [43]. This is usually measured using the gini index for classification and the Mean Squared Error (MSE) for regression [41, 42, 43]:

$$Gini\ I(t) = 1 - \sum_{i=1}^{c} p(i|t)^2 \qquad MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

The depth of a tree controls how well it can learn the data and, as such, the bias-variance trade-off. Specifically, deeper trees lead to a reduction of bias but an increase in variance. Despite being a very intuitive model, trees are usually used as part of an ensemble as in random forests, improving performance at the cost of interpretability [43].

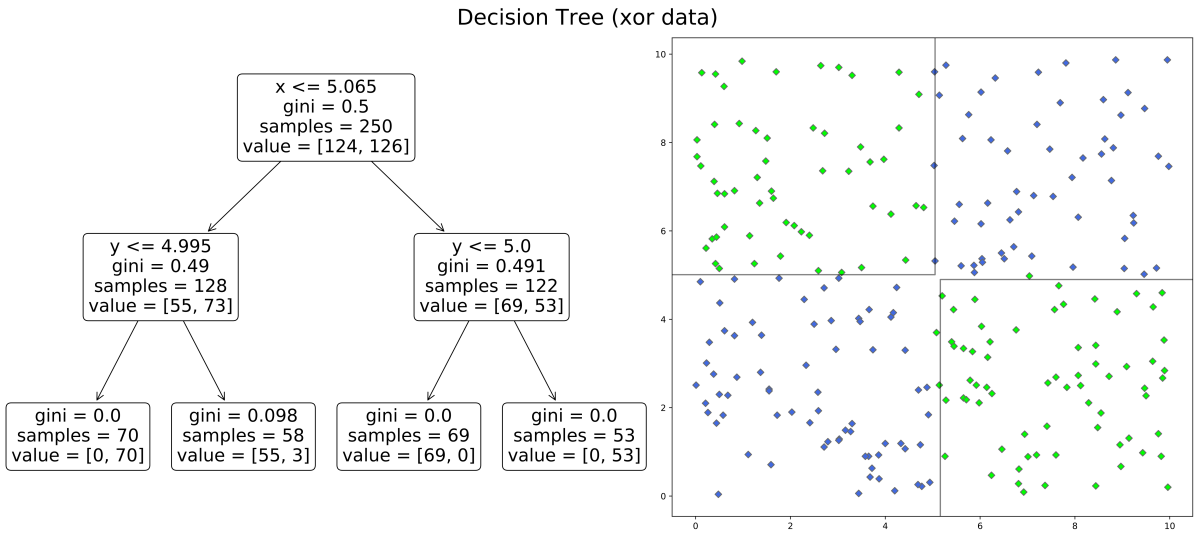


Figure 9: Decision boundary of a decision tree fitted on XOR data and its internal structure. Each leaf in the tree defines a discrete region of the input space, with samples within receiving the same label.

Random Forests (RFs) combine many trees under a single model, with the core idea being that combining the predictions of many trees reduces variance. Specifically, it uses trees that weak learners and by combining their predictions produces a model with a good bias-variance trade-off [43].

Each tree in this ensemble is trained using a dataset made by repeatedly sampling, with replacement, the original training dataset, a process also known as bootstrapping. Furthermore, during construction of the individual trees, each rule is chosen considering only a random subset of the total features. This allows for the trees to be uncorrelated with each other even when a strong predictor exists for the data. The output of the model is then taken as the majority vote, in classification tasks, or the average prediction, in regression tasks, of all the trees [43].

In practice, RFs are quite robust to noise from individual trees and, as such, large numbers of trees in the ensemble don't usually lead to overfitting. Nevertheless, this hyperparameter does affect computation time and, as such, is typically increased, within reason, until it doesn't improve predictions [41, 43].

3.3.3 *Artificial Neural networks*

Artificial Neural Networks (ANNs) were originally inspired in biology, but over time developed into models that bear little resemblance to their natural counterparts [44]. The neuron, also referred to as perceptron, is the basic unit of ANNs. It learns a weighted sum of its inputs (with an added bias), or in other words, performs a linear transformation of the input [42]. This can be enhanced by passing the output through

non-linear activation functions, for example the unit step function, which enables a perceptron to perform classification [42]. Figure 10 (left) illustrates this model.

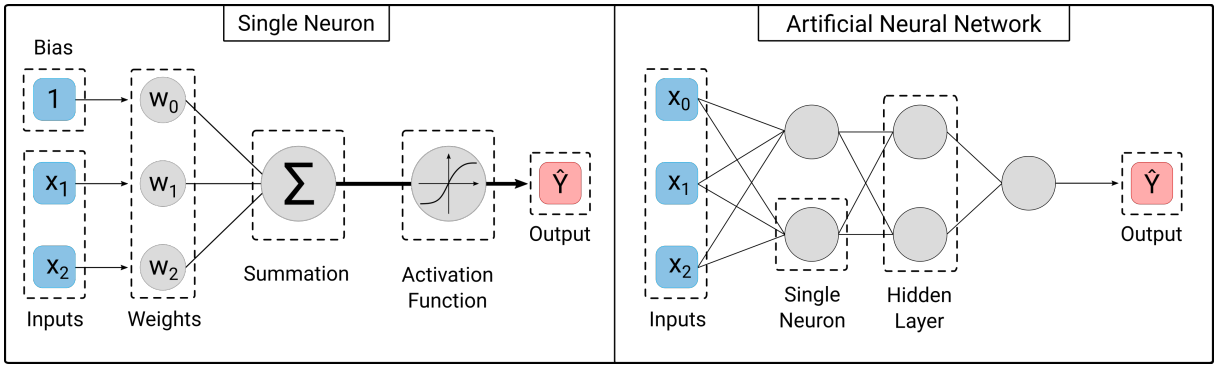


Figure 10: **Left**: illustration of a single neuron performing a weighted sum of the inputs and applying a non-linear activation function to the result. **Right**: small ANN with 5 neurons organized into 3 layers. Namely a input layer with 2 units, a hidden layer with 2 units and a output layer with a single unit.

In feedforward ANNs, neurons are stacked into multiple layers such that the output of a neuron in a layer serves as input for neurons in the next layer, Figure 10 (right). The intermediate layer, between the input and the output layer, is known as a hidden layer and the whole model is frequently termed Multi Layer Perceptron (MLP) [41]. The staking of consecutive non-linear activation functions enables the model to learn non-linear transformations of its inputs and, therefore, approximate non-linear functions [42, 44]. The architecture of an ANN defines the number of neurons, their organization into separate layers and the connections between them. Particularly, the main concern when defining an ANNs architecture is its depth, how many layers it has, and its width, how many neurons each layer has. Furthermore, the number of neurons in the ANN roughly establishes the model's capacity. The width and depth are hyperparameters of the model and are usually chosen by monitoring the performance on a validation set [44].

ANNs learning process happens in three steps [41, 44]:

- First, the inputs are feed into the input layer, where each unit computes a weighted sum of the inputs and applies an activation function to the result. The output of each unit is then fed into the next layer and the process repeated on the following layers. This first step is also termed **forward propagation** because the information is said to flow forward, from input to output.
- In the second step, an error is measured from the output of the last layer. This is accomplished through a loss function that embodies the objective.
- In the last step, the error is used to update the weights of the network, usually through gradient descent. Here, starting at the output layer and progressing until the input layer, the gradient of the error with respect to the weights of each layer is computed. The weights of the network are then updated by taking a step in the opposite direction of the gradient. This third step is also known as **backpropagation** because the information is said to flow backwards, from output to input.

3.4 Deep Learning

Deep learning is a particular case of ANNs that uses Deep Neural Networks (DNNs). Over the last decade, deep learning has proven successful in multiple fields such as computer vision, speech recognition and translation, frequently pushing the state-of-the-art forward and surpassing other machine learning approaches [44, 45]. This field is undergoing intense research to improve current methods and apply them to new problems.

3.4.1 *Artificial Neural Networks and Deep Learning*

Deep learning refers to the use of artificial neural networks with multiple hidden layers or, in other words, using deeper neural networks. The main motivation for using deeper networks is that such architectures usually learn the desired functions employing fewer parameters than their shallower counterparts [44]. A common intuition to this is that successive layers can learn higher level abstractions of the inputs.

The emergence of deep learning techniques may be attributed to multiple factors. The appearance of large scale data-sets, improvements to the training algorithms and advancing computational hardware combined to enable the ongoing success of deep learning [44, 45].

3.4.2 *Architectures*

The structure of the network can be made to reflect some prior knowledge of the task it is intended to perform. This prior knowledge consists of various characteristics of the task that are independent of the training dataset used [42]. As such, different architectures will encode different assumptions about the data. Generally, these assumptions lead to limiting the capacity of the model, however they can also greatly improve performance when applied to the correct data [45]. There are currently a number of different architectures that have proven useful in deep learning.

Densely Connected Networks (DCNs) do not encode any assumptions about the task, attempting instead to find interactions among any combination of features, limited mainly by the number of neurons in the network. They are termed dense because each unit in a layer is connected to all the units of the next layer, as depicted in Figure 11. Common uses of DCNs are vector data, where no structure is discernible, categorical data and as the output layer of most deep networks. When defining a dense layer, a single fully connected layer, the main consideration is its number of units [44, 45].

Convolutional Neural Networks (CNNs) assume a grid-like structure in the input data, that is, a spatial arrangement where local relations between neighbouring features exists. This structure can be 1 dimensional, as with time-series, 2 dimensional, as with images, 3 dimensional, as with videos, or in higher dimensions. A convolutional layer, a single layer of a CNN, applies local transformations across

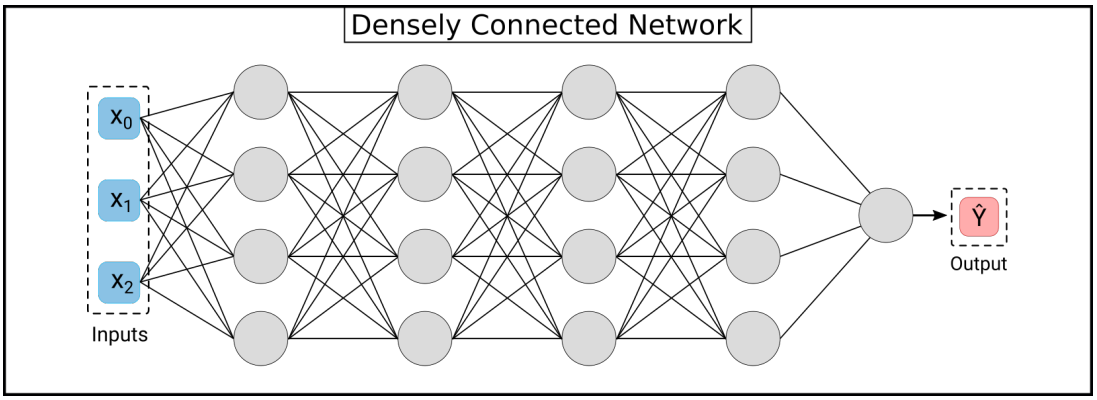


Figure 11: DCN with 17 units organized into 5 layers, 4 of which are hidden layers. Dense connections between each layer means that with only 17 units the network already has 81 learnable parameters (12 input + 48 hidden + 4 output + 17 bias).

the whole input. This is accomplished by applying a sliding window of weights to the input and producing locally weighted sums. As Figure 12 shows, this window is termed kernel or filter and is typically much smaller than the input. Unlike DCNs, where a weight exists for each input, a convolutional layer applies each weight to every input, reducing the total number of parameters to be learned and making it more efficient when applied to the correct data. When defining a convolutional layer the main considerations are the number of filters and their size [44, 45].

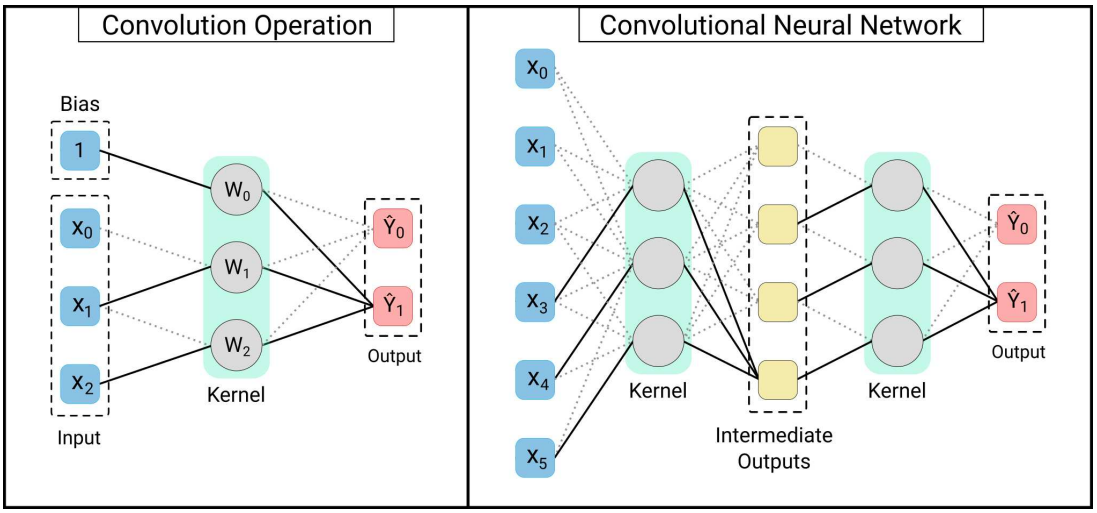


Figure 12: **Left**: Application of a kernel (size 2) to a input with 3 features. The dotted lines illustrate the sliding window effect where the same weight is applyed to different parts of the input. **Right**: CNN with 2 layers, each composed of a single kernel of size 3. The dimention of the output is smaller than its corresponding input due to a border effect. This can be corrected with the use of padding.

Recurrent Neural Networks (RNNs) assume a sequential structure in the data, one where a sample is composed of a set of steps, each depending on previous steps and also influencing later steps. Each part of the input sequence is also termed time-step. A recurrent layer, single layer of a RNN, implements this assumption by processing an input in a sequential manner, one step at a time, and by introducing a recurrent loop, a connection that carries the output from previous steps into the current step, Figure 13

(left). This translates to performing a weighted sum of the features of each time-step and the output of the previous step. These weights are usually divided into a set of weights parameterizing the inputs (input weights) and another set parameterizing the state (recurrent weights). As illustrated in Figure 13 (right), a recurrent layer can be composed of more than one unit, in which case, at each step a unit receives the input of that step and also the output of all the units of its layer from the previous step. In other words, the units in a layer have dense recurrent connections between steps. These networks can be unfolded and thought of as a long chain, applying the same parameters to the input at each step.

However, as the number of steps increases, RNNs can suffer from vanishing or exploding gradient in the backpropagation step, where the gradient goes to 0 or infinity, respectively. This impairs the training process, and for longer sequences makes learning of long term dependencies extremely difficult. This problem can be improved upon by the introduction of specialised units such as Gated Recurrent Units (GRU) or LSTM. These units introduce gates, learnable parameters controlling the flow of information throughout the steps [44, 45].

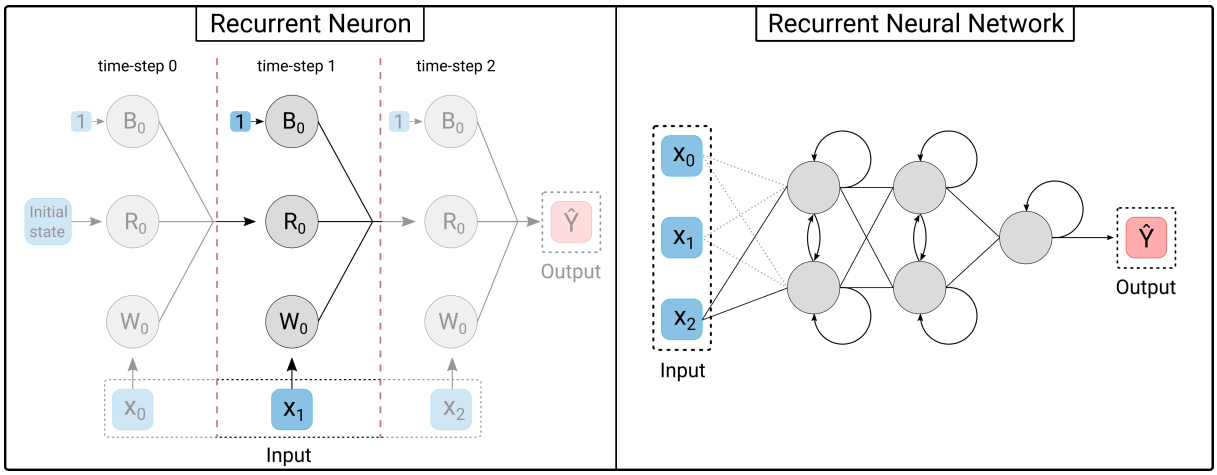


Figure 13: **Left**: single recurrent unit unfolded over 3 time-steps. It is composed of 3 weights, a input weight (W_0), a recurrent weight (R_0) and a bias weight (B_0). The unit applies the same weights for all time-steps. **Right**: RNN with 5 units over 3 layers, single hidden layer. In each layer the units have dense recurrent connections, meaning the output of a unit in a step flows to all units of the layer in the next step. It has 22 parameters (8 input, 9 recurrent, 5 bias)

Long Short-Term Memory (LSTM) introduce three gates controlling the flow of information and a cell state, a connection that can carry long-term or short-term dependencies. At each step, the cell state is computed by applying two transformations to the previous state. These transformations are termed the forget gate, meant to allow the cell to discard information, and the input gate, meant to allow the cell to store information. The output of the cell for that time-step is obtained by applying a final transformation to the new state. This transformation is termed the output gate and is meant to allow the cell to apply its memory to the current output. In practice, these interpretations don't necessarily hold as the operation these gates perform is learned and depends on the inputs. Particularly, each gate corresponds to a set of learnable parameters applied to the input and recurrent connections of the respective time-step. As Figure 14 shows, an LSTM effectively receives 3 inputs at each step: the features of the input at the time-step,

the output of all units from the previous step and its own state from the previous step [44, 45].

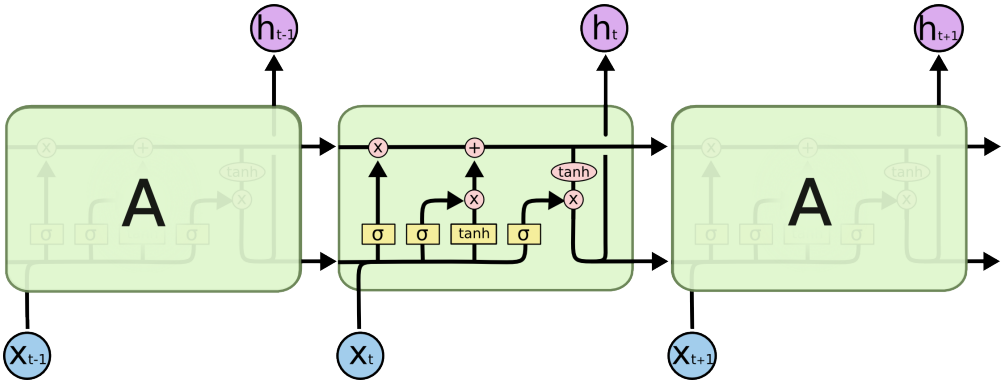


Figure 14: LSTM cell unfolded over 3 time-steps. At each step the cell receives 3 inputs and computes 2 outputs, the new cell state alongside the time-step output. Reproduced from the tensorflow repository [46]

Gated Recurrent Units (GRU), Figure 15, do not employ a cell state and only apply two gates to the output of the previous step. These are, similarly to an LSTM, a reset gate and an update gate. The reset gate is applied to the previous output, intending to allow the unit to discard previous information. The update gate is applied both to the previous output and a newly computed preliminary output, which then combine to form the output of the unit at that time-step. This gate is intended to control how much information from the previous output flows into the next. However, the reasoning about these interpretations made earlier still holds, with the function each gate performs being actually learned and depending on the data. The GRU was inspired in the LSTM but intended to be simpler in both implementation and computation, providing most of the gating information flow benefits while requiring fewer parameters [44, 47].

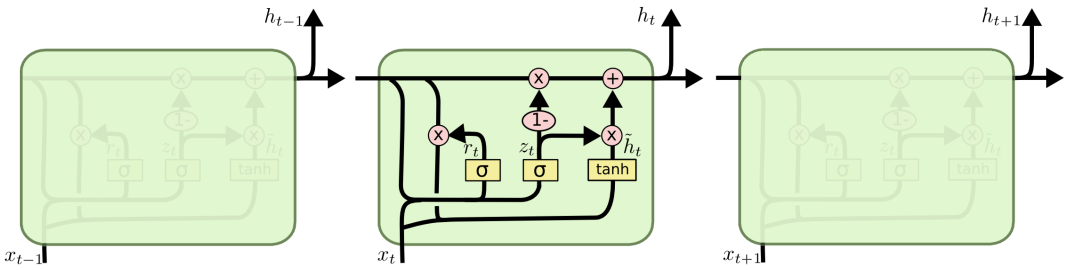


Figure 15: GRU cell unfolded over 3 time-steps. Unlike LSTMs no cell state is present, instead long term dependencies are encoded in the output of each step. Adapted from[48]

Generative Adversarial Network (GAN) define a pair of networks, a generator and a discriminator, trained in competition with each other, Figure 16 (bottom-left). The generator is intended to transform random noise into real looking data and is trained to maximize the synthetic samples classified as real by the discriminator. Meanwhile, the discriminator is trained to better discern between generated and real data. The training framework resembles a competition, with both networks constantly improving and adapting to each other [44, 45, 49].

Autoencoders (AEs) are neural networks trained to copy their input into the output, with restrictions imposed to avoid learning the identity function. As Figure 16 (top-left) shows, they are usually thought of as two separate parts, an encoder that transforms the input into a more compact latent state, and a decoder that reconstructs the input from this representation. Both are trained together to minimize the information lost from reconstructing. A common way to restrict an AE is to constrain the latent space to a lower dimension than the input, creating an information bottleneck that forces the AE to extract the input's most significant patterns. These are termed undercomplete AEs. An alternative is to alter the loss function by adding a regularization term forcing the latent state to be sparse as in sparse AEs, or by perturbing the input with noise and expecting the original unperturbed data, as in denoising AEs. AEs learn in an unsupervised manner where each input serves as its own target [44, 45].

Variational Autoencoders (VAEs) are a special type of AEs that assume that the data was sampled from an arbitrary statistical distribution. The concept of an encoder and decoder attempting to recreate an input remains. However, as depicted in Figure 16 (top-right), in a VAE the encoder transforms its input into the parameters of a multidimensional statistical distribution, namely a set of means and standard deviations. A sampling then occurs, where a point is drawn from the encoded distribution and fed into the decoder, which reconstructs it into the original input. The objective function used for training consists of penalizing reconstruction errors and restricting the parameters encoded to be close to a prior distribution, usually the normal $\mathrm{N}(0, I)$ where I is the identity matrix. This stochastic process as well as constraining the encoded parameters close to those of a normal distribution helps in forming a useful latent space. [45, 50].

Adversarial Autoencoders (AAEs), illustrated in Figure 16 (bottom-right), are an alternative to VAEs that employs adversarial training for structuring the latent space. Particularly, in AAEs the encoder transforms its input into a single point in the latent space. A discriminator network then attempts to differentiate between samples of a prior statistical distribution and encoded points. As such, the encoder can also be viewed as a generator engaged in a competition with the discriminator, ultimately balancing between the reconstruction and adversarial error [51].

3.4.3 *Training of Deep Networks*

Training of deep neural networks follows the same 3 steps outlined for ANNs. However, some improvements have been made to the training algorithms that enhance both the computation time required for training and the final model's generalization capability. These improvements are generally also applicable to shallow networks.

As detailed for ANNs, each sample is processed by the network independently, however the measurement of the error and its subsequent back-propagation can be performed for any number of observations at a time.

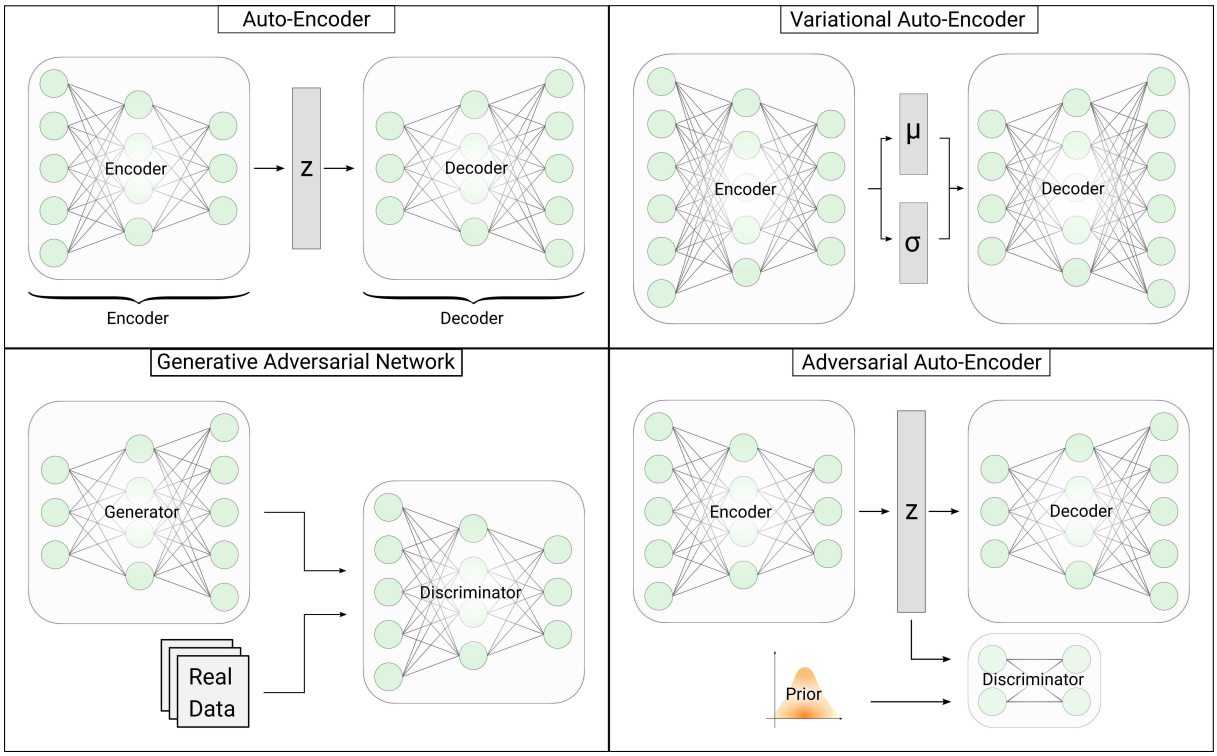


Figure 16: **Top-left**: AE composed by a encoder and a decoder, each with 2 layers. The latent space is restricted to a lower dimension than the original input. In practice, this latent state corresponds to a single vector; **Top-right**: VAE where the input is encoded to the parameters of a statistical distribution, namely the means (μ) and standard deviation (σ). In practice, these correspond to 2 vectors which, on the sampling step, are interpreted as a set of means and standard deviations; **Bottom-left**: GAN composed by a generator (G) and a discriminator (D). Training seeks not a minimum but a useful equilibrium between the generator and the discriminator. **Bottom-right**: AAE where the attached discriminator must discern between encoded points and samples drawn from a prior statistical distribution.

In **batch gradient descent**, the model first processes the full training set and the error is averaged over all outputs. This can be computationally expensive as it requires that the full dataset be evaluated for a single update [44].

In **minibatch stochastic gradient descent**, the model processes small batches randomly sampled from the training set, and the error is averaged over only those samples. This reduces computation cost while improving generalization as the stochastic nature of the sampling introduces noise into the training process, resulting in a regularizing effect. Here, the size of the minibatch is usually a power of 2 (64, 128, 256, ...) and commonly referred to as batch size [44, 45].

Adaptive learning rates is a method for speeding up the training process. Here, a separate learning rate is kept for each parameter enabling the model to take large leaps in some parameters and only small steps in others. Furthermore, these learning rates are continually updated based on their past gradients, that is, if a parameter always advances in the same direction, it may converge faster with a bigger learning rate. Conversely, if a parameter constantly shifts the direction of change it may benefit from a lower learning rate. Taken together, these allow the model to adapt its learning process and, as such, reduce

the time required for training [44].

Momentum is a different approach to improving the training process where the step does not solely depend on the current gradient and learning rate but also on a velocity. Particularly, this velocity corresponds to an exponentially decaying average of past gradients, meaning that more recent gradients hold more importance than distant ones. Momentum can be intuitively understood as its homonym, physical momentum. Here, the optimization process can be thought of as a small ball moving in the loss surface. As it descends, it gains velocity that may later enable it to climb out of local minima [44].

Transfer Learning is a training procedure where the model first learns to perform a task similar to the one desired, but for which bigger datasets exist, before being fine-tuned on the intended data. This approach assumes that several of the underlying attributes learned on the first set are transferable to the second set. As such, when only small datasets are available for the intended task, larger similar datasets can be leveraged to facilitate the learning process. For example, different tasks may share the same output structure but different input distributions, as with speech recognition. Here, the model needs to output valid sentences, but its input varies with different speakers. This means that the task learned by the last layers of a network will be similar independent of the speaker used for training. Therefore, these can be re-purposed from one model to the next, only requiring that the lower layers be trained with the different speaker [44].

Teacher Forcing is a procedure for easing the training of the output layer in RNNs where, at each step, the units in the last layer receive the ground truth instead of the output of the previous step. This effectively replaces the recurrent connections of the layer, enabling it to focus on matching its output to the expected target. However, this can lead the layer to become over reliant on this signal and not encode information in the output of a step that would be relevant to later steps. If the network is then used in a free-running mode, with proper recurrent connections in the output layer, its inputs may differ significantly from those experienced during training. A solution is to alternate between the teacher-forced and free-running mode, enabling it to learn to encode relevant information in its output while also enabling it focus on matching the target output. In a curriculum learning approach, this can be made to start the training on a teacher-forced mode and gradually transition into fully free-running mode [44].

3.4.4 *Regularizing Deep Networks*

Improving the generalization capability of deep models can be achieved by introducing regularization techniques. Three of the most prominent are weight decay, dropout and batch normalization.

Weight regularization, also known as **weight decay**, introduces a term into the loss function penalizing large weights, driving the model towards being simpler, less complex. A hyperparameter, α, controls the regularization strength and, therefore, the bias-variance trade-off. There are two popular approaches,

the L1 and the L2 norm. The L2 norm, the most popular of the two, penalizes weights by the norm of their square [44]. It encourages reducing the value of the weights but not setting them to zero, because it offers diminishing returns for values approaching zero. The L1 norm penalizes weights by the norm of their absolute value. It encourages sparsity as it rewards setting parameters to absolute zero [44, 45].

Dropout is a regularization technique where the output of a number of units is set to zero, effectively removing them from the network. The proportion of units to be dropped is set *a priori* and the units effectively removed change with each batch. That is, for each sample, a mask is randomly created based on the set proportion, and used to remove the units. When using stochastic gradient descent with dropout, a different mask is made per minibatch. As such, this can be thought of as a bagging process where a number of different models, differing on which units were removed, are trained on bootstrapped samples. Furthermore, these models also share units among each other encouraging each unit to not be particularly reliant on any other unit. This process is applied during training and when testing no units are dropped, instead the weights are scaled-down depending on the proportion of dropout used, ensuring that the output of each unit is similar to when using dropout [44, 45, 52].

Batch normalization is a technique where the inputs to a layer are normalized, fixing their mean and variance. During training, as the parameters are updated, the distribution of inputs each layer receives changes. This impairs training as it forces the network to not only learn the intended task but also continuously adapt to the changing distribution of activations. Batch normalization addresses this by normalizing each feature of a layers input based on the current minibatch mean and variance. However, as this process reduces the models capacity, two new learnable parameters are introduced for each input feature of the layer, β and γ which respectively shift and scale the normalized values. The full process, normalizing followed by shifting and scaling, effectively means the layer doesn't have to adapt to the continuously updating parameters preceding it. Instead, the inputs follow a distribution defined by the newly introduced β and γ, facilitating training [44, 45].

3.5 Software / Packages

There are currently various software packages and libraries implementing relevant tools frequently used in machine learning and in deep learning. These libraries help with the reproducibility of published work and also ease the development of new approaches. Some examples are:

Scikit-Learn [53] - Machine learning module for python, implements a broad toolkit for developing machine learning models.

Tensorflow [46] - Symbolic math library available in python, implementing many useful operations on tensors. Provides the most popular back end for deep learning computation.

Pytorch [54] - Deep learning module for python, implementing various popular deep learning architectures. Allows the easy development of custom deep learning architectures.

Keras [55] - Deep learning module for python built over tensorflow, providing a high level abstraction of deep learning. Its focus is on allowing fast experimentation.

4

DEEP LEARNING FOR *DE NOVO* MOLECULAR DESIGN

In the *de novo* molecular design the aim is to create new chemical entities with some desired properties. These properties may be easily quantifiable, such as molecular weight, or somewhat abstract such as being non-toxic. This is an inherently difficult task owing to the immense search space of around 10^{33} - 10^{80} feasible molecules of which only a very small fraction have the desired properties [2]. As such, *de novo* molecular design was for many years, and mostly remains, a process of almost exclusive trial and error with human expert knowledge and intuition about chemistry playing a major role [3]. Meanwhile, the high costs associated with developing new molecules, reaching 2.8 billion dollars for a single compound, has also led to the implementation of computational tools capable of assisting the process. These have proven valuable and have found wide usage in practical applications [1, 3].

4.1 Approaches other than Deep Learning

Enumeration is an approach where of all possible molecules that conform to valency rules and don't include chemically unstable functional groups are listed. An example is the chemical space project, where this technique was employed to generate 166 billion molecules [31, 56].

In **reaction-based** *de novo* design, a set of known chemical reactions are used to combine (*in silico*) various readily available building block into new molecules. This process can then be guided by a similarity criterion to a known molecule of interest, giving rise to large numbers of new similar molecules that are synthetically plausible [56, 57].

After their creation, these large virtual libraries are searched for molecules of interest. This **virtual screening** of molecules is usually accomplished based on the target, such as in docking, or on known ligands, such as in similarity searching. Docking refers to a process of fitting a molecule to the binding site of a given target whose 3D structure is known. This is usually attained by scoring different poses, spatial orientations, of a molecule relative to its target. The score is calculated by a scoring function, usually based on predicting the change in Gibbs free energy. In a similarity search, the similarity between a molecule and a set of molecules that are known to be active is determined. A common approach is to compute the Tanimoto coefficient between the fingerprints of each molecule [22, 58].

Evolutionary Algorithms (EAs) have also been applied to the design of new molecules since the 1990s [59]. As a recent example, AutoGrow4 [60] uses an EA to create new predicted ligands. At each iteration,

new molecules are created using a mutation operator, that performs an *in silico* chemical reaction, or a crossover operator that merges two compounds into a new one by randomly combining their decorating moieties. Grammatical Evolution working over string representations, as well as evolving molecular graphs, provides alternative approaches that enable EAs to generate novel compounds targeting desired properties [61, 62].

4.2 Generating Molecules with Deep Learning

There have been several approaches to applying generative DL to molecular generation, mainly differing on the chosen molecular representation used. As such, usually more than one method surfaced for generating each of the main representations discussed in Chapter 2.2.

Borrowing from the natural language processing field, molecules can be generated as sequences, such as SMILES, by using RNNs. Specifically, when using RNNs as a generative model, each token in the string is encoded as a one-hot vector, and the network is trained to predict the next character in the sequence. The generation of new data is achieved by running the network auto-regressively, that is, using its output as the input for the next time-step. This process is usually seeded with a special start token and the generation of a molecule ends when a special stop token is sampled. These two tokens are also respectively prefixed and appended to each molecule during training, Figure 17 illustrates the generative process.

Several research groups have employed this method with a stacked RNN, usually with LSTM cells, leading to good rates of validity, novelty and diversity [7, 63, 64, 65]. More complex architectures such as VAEs and GANs have also been employed to generate molecules as strings, however these also employ a RNN for the sequence generation process, either as the decoder or the generator [10, 66, 67].

Although very versatile in representing compounds, generating molecules represented as InChI in DL has not been successful. This was first observed by Gómez-Bombarelli et al.[66] who attributed this result to the added complexity of the InChI syntax, when compared with SMILES. A later work by Winter et al.[12] also noted inferior performance when translating from SMILES to InChI, reporting that the model failed to learn, identifying the same probable cause. As such, thus far most methods have employed the SMILES representation.

Despite some limitations of sequence-based approaches, such as the need to learn a complex syntax and the mismatch between the edit distance of two SMILES and the underlying molecular similarity, these methods have produced impressive results including some instances of experimental validation of the generated molecules [8, 9, 68, 69]. Furthermore, two new sequence based molecular representations have recently been introduced. O'Boyle et al.[70] proposed DeepSMILES, an adaptation of the SMILES syntax focusing on two of the major causes of generation of invalid SMILES, unmatched ring and parentheses closures. This is addressed by using a single symbol to indicate rings and by denoting branches solely using closing parenthesis. Krenn et al. [71] introduced a linear notation for constrained graphs, termed SELF-referencIng Embedded Strings (SELFIES). It is capable of enforcing the generation of syntactically and semantically valid graphs and is readily translated to and from them. This team compared its perfor-

mance against SMILES, DeepSMILES and Kusner et al. [72] GrammarVAE, reporting improvements on validity, reconstruction accuracy, and diversity of generated molecules.

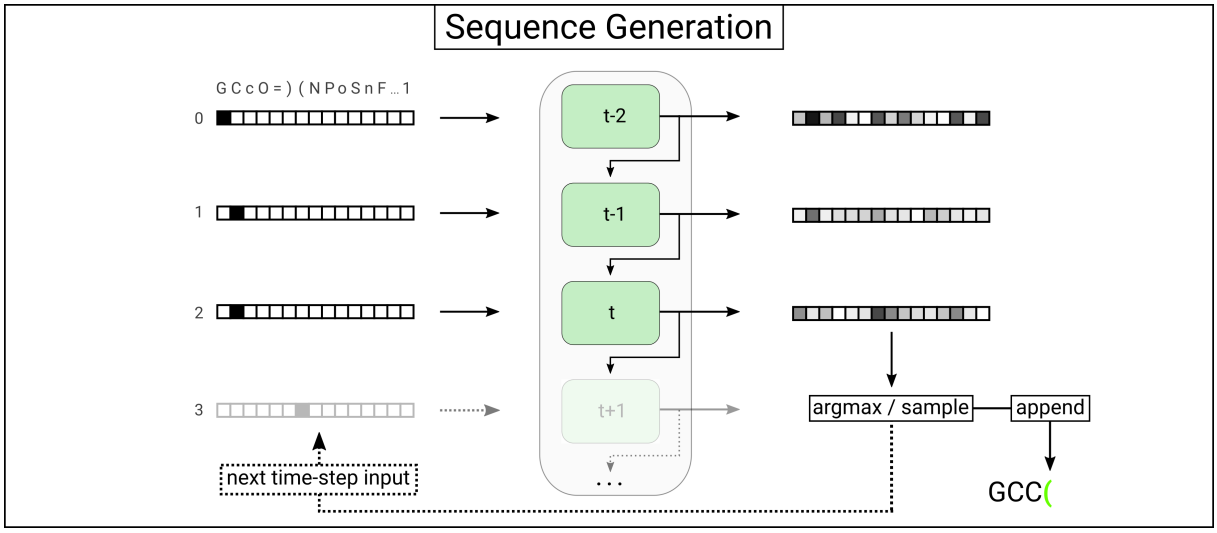


Figure 17: Three layer RNN, unfolded over 4 time-steps. In auto-regressive sequence generation, the process is started with a special start token, here 'G'. The model then predicts the next token, which is sampled and used as input for the next step. Generation ends when a stop token is predicted.

In an attempt to present molecules in a more natural and intuitive form, several methods have been proposed to directly generate molecular graphs. The generation process can be modeled as a sequence of decisions that progressively builds a graph. As Figure 18 describes, this approach usually employs multiple networks with specific functions, such as adding new nodes or adding edges between existing ones. In this paradigm, Li et al.[73] used two Graph Neural Networks (GNNs) to build graphs, one deciding whether or not to add a new node followed by another network deciding whether, and where, to add new edges between the existing nodes. Liu et al.[74] used a similar generation procedure, employing Gated Graph Neural Networks (GGNNs) to build a VAE. The decoding process starts by using a linear classifier to add attributes to a fixed number of unconnected nodes. Edges between these nodes are then progressively added with two densely connected networks, one deciding the target node and the other the type of edge. At each step, the attributes of the nodes are updated with a GGNN and the process ends when a special stop node is selected. Furthermore, invalid operations can be masked during the generation allowing, for example, the enforcement of valency rules. More recently, Mercado et al.[75] compared six GNN architectures coupled with a tiered MLP for sequentially building molecular graphs, reporting that GGNNs showed the best performance for both speed and quality of the generated structures.

An alternative is to generate graphs in a one-shot fashion by directly outputting an adjacency matrix and corresponding attribute tensors. De Cao and Kipf[76] applied this approach with MolGAN, a GAN whose generator outputs probabilities over the adjacency matrix and the annotation matrix of a molecular graph. Following a similar approach, Simonovsky and Komodakis[77] employed a VAE with a decoder that outputs three probability distributions, one over the adjacency matrix, one over the edge attribute tensor, and another over node attribute tensor. Furthermore, Ma et al.[78] proposed a regularization scheme capable of enforcing validity constraints to graphs generated in the same manner, significantly improving

the validity of generated molecules.

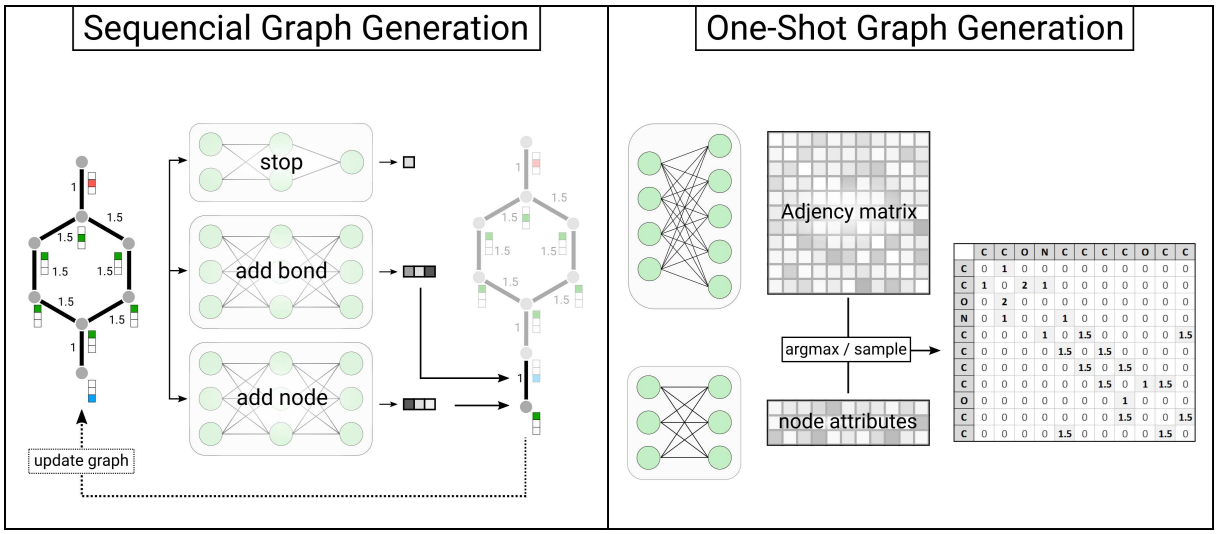


Figure 18: **Left**: In sequential graph generation, a graph is built by evaluating a current partial graph, adding a node/edge and repeating until the network outputs a stop signal. **Right**: In the one-shot generation of graphs, probabilities over the full adjacency matrix and node/edge attribute tensors are produced. The graph is then obtained by taking a sample or the *argmax* of these outputs.

Molecules are ultimately three-dimensional objects, with electron clouds surrounding their atoms and multiple possible spatial arrangements or stereoisomers. By generating molecules as sequences or graphs, important information is omitted, possibly hindering *de novo* design. Furthermore, determining the relevant conformations is not a trivial problem, as even small molecules can have many possible conformations [79]. As such, some attempts have been made towards generating molecules as three-dimensional entities. Skalic et al.[80] proposed to generate voxelized molecular shapes with a VAE and then caption them into SMILES with a separate network. A different approach was proposed by Gebauer et al.[81] where molecules are generated as point sets, iteratively built based on the pairwise distance to previously placed points/atoms (Figure 19). Figure 19 outlines the general process of these two last approaches.

4.3 Evaluating generative models

Given the rapid growth this field is experiencing, the development of systematic and robust methods to assess the performance of novel approaches is essential to help guide future work.

Aiming to help address this, Preuer et al.[82] introduced the Fréchet ChemNet Distance (FCD), a metric for comparing the generated molecules against the training dataset. This score measures the distance between the hidden representations of the two sets of molecules in 'ChemNet', a recent multi-task network for predicting biological activities. As this network was trained to predict the bioactivities of about 6000 assays, the team proposes that FCD combines into a single metric a multitude of important molecular features, which is therefore useful to evaluate generative models.

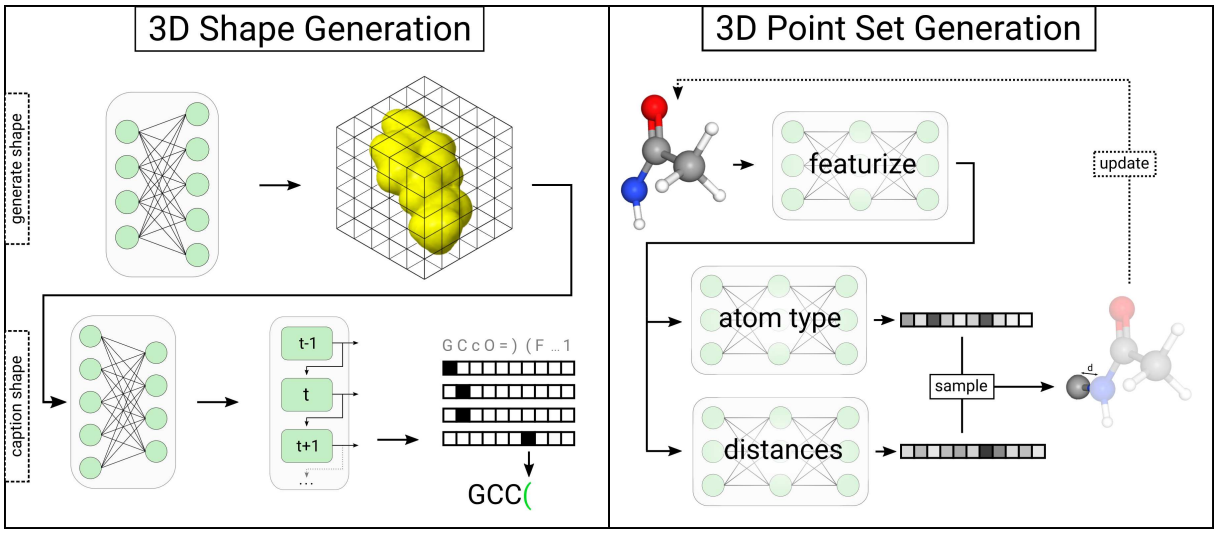


Figure 19: **Left**:General procedure for the generation of 3D shapes as proposed by Skalic et al. [80]. The convolutional decoder of a VAE is used to produce a 3D molecular shape which is converted to SMILES by a captioning network. **Right**: General process for generating molecules as 3D point sets, proposed by Gebauer et al.[81]. It is conceptually similar to the sequential graph generation, operating on point sets with an internal coordinate system.

Arús-Pous et al.[83] proposed a method to evaluate how well a generative model learns to cover the relevant chemical space. According to the team, this can be accomplished by training the model on a fraction of a large enumerated dataset, such as GDB-13, and then tracking the percentage of the total dataset the model can recover, how uniform the coverage is, and also whether it generates molecules outside the dataset. These results can then be compared to an ideal model, directly sampling the dataset, that serves as an upper bound for performance. Furthermore, the team introduced a method for evaluating the quality of the training process by comparing the negative log likelihood of the sampled, training and evaluation sets throughout the training process.

Brown et al.[84] introduced GuacaMol, a framework for benchmarking models for *de novo* molecular design. This framework was divided into two main sets of benchmarks, distribution-learning and goal-directed generation, aiming to emulate the two main use cases of these models. The first set measures how well the models learn to generate new molecules, using validity, uniqueness and novelty rates, and also whether they match the properties of the training dataset, measuring the FCD and the divergence in the distribution of a variety of physicochemical descriptors. The second set evaluates the targeted generation performance, including benchmarks such as optimizing given molecular features or properties, generating molecules similar to a target compound or recover a specific target molecules. In addition, the team also included a benchmark for assessing molecular quality, leveraging rule sets for building high-throughput screening libraries. Lastly, a standardized dataset alongside a number of baselines were also released with this framework to help compare novel approaches.

Polykovskiy et al.[85] proposed the benchmarking framework MOSES, for evaluating the distribution learning performance of generative models. To assess whether the models can generate new molecules, it measures the validity, uniqueness and novelty rates along with the internal diversity and the fraction of generated molecules that pass a set of structural filters for molecular quality. The framework also provides

a set of metrics meant to evaluate how well the model learned features of the training dataset. For this purpose, it presents the FCD, the distance between the distribution of various physicochemical properties, the Tanimoto Similarity to a nearest neighbor, the cosine similarity of Bemis–Murcko scaffolds, and also the cosine similarity of BRICS fragments. These last two metrics help compare molecules at a substructure level, while the first three help to capture more abstract chemical and biological similarities. The team also released various useful baselines, as well as a standardized dataset with a recommended train, test and scaffold test split.

Building on these early works, other approaches to evaluating molecular generative models have continued to be developed. Specifically, Renz et al.[86] highlighted some shortcomings of currently used evaluation metrics. Specifically, the team details how a trivial model can excel in distribution-learning benchmarks and also how goal-directed generation can exploit biases in the scoring functions, producing compounds with high scores but of little practical use. Cieplinski et al.[87] proposed to better represent real discovery problems by using docking as a benchmark of the different methods of goal-directed generation. Zhang et al.[88] improved on their earlier work, measuring the coverage of chemical space by generative models[83], by also evaluating the coverage of functional groups and ring systems. Furthermore, the team provided results for various, recently introduced, generative model architectures allowing for their comparison as well as providing useful baselines for future works.

4.4 Generating Compounds of Interest

The automatic generation of novel molecules usually targets specific properties and characteristics, such as solubility or bioactivity. As such, the ability to create not just new, but also focused, molecules is of interest. Table 2 summarizes state-of-the-art approaches for targeted compound design, further described in this section.

4.4.1 *Screening*

Testing large numbers of compounds to see if they show evidence of having desired properties is usually one of the first steps in the drug discovery pipeline. This usually entails a virtual screening for bio-activity either based on a target, as in docking, or on known ligands, as in ML classifiers and similarity searching [58]. This process can be used on its own or after other methods of biasing the generation process. Docking refers to a process of fitting a molecule to the binding site of a given target whose 3D structure is known. This is usually attained by scoring different poses (spatial orientations) of a molecule relative to its target. The score is calculated by a scoring function, usually hinged on predictive changes in Gibbs free energy [22].

Yuan et al.[68], for example, docked generated molecules against VEGFR-2, a mediator of the VEGF angiogenesis pathway, to choose compounds to be synthesized. Similarly, Polykovskiy et al.[69] docked molecules against Janus kinase 2 and 3 as part of the selection process for synthesis.

Table 2: Methods for the directed generation of molecules

Method		Representation	Architecture	Reference
Transfer Learning		SMILES	Stacked RNN	[7, 8, 9, 64, 89, 90]
			-	[91]
		Graph	GNN	[92]
		3D point sets	SchNet + 2 MLP	[81]
Reinforcement Learning	Pre-train + RL	SMILES	Stacked RNN	[7, 63, 90, 93]
			VAE	[94]
		Graph	Two RNNs	[95]
	Adversarial + RL	SMILES	GAN	[10, 96, 97, 98]
		Graph	GNNs	[99, 100]
			GAN	[76]
Latent Space Navigation	Bayesian Optimization	SMILES	VAE	[66, 101]
			AAE and VAE	[11]
		SMILES (production rules)	VAE	[72, 102]
		Graph (junction trees)	VAE	[103]
		Graph	VAE	[104]
	Gradient Ascent	Graph (junction trees)	VAE	[103]
		Graph	VAE	[74, 105]
	DL Model	SMILES	GAN + AE[106]	[107]
		Graph (junction trees)	CycleGAN + VAE[103]	[108]
	GA	SMILES	AE	[109]
	PSO	SMILES	AE	[110]
	CLaSS	SMILES	VAE	[111]
Conditioned	Conditioned	SMILES	VAE	[67]
			Stacked RNN	[112]
			Two AAEs	[113]
		SMILES (production rules)	Two GANs	[114]
		SELFIES	VAE	[115]
		Graph	GNNs	[73, 92]
		Graph	VAE	[77]
		Graph (junction trees)	VAE	[116]
		3D shape	VAE + RNN	[80]
			VAE + GAN	[117]
	Semi-supervised	SMILES	VAE	[118]
			AAE	[69]
		Graph (scaffold extention)	VAE	[119]

An alternative to predict how well a molecule may bind to a receptor is through the use of supervised machine learning classifiers trained to distinguish between known actives and not actives. An example of this approach is the work done by Olivecrona et al.[63] where a support vector machine was trained to predict activity towards the Dopamine Receptor D2 (DRD2).

In a similarity search, the similarity between a molecule and a set of molecules that are known to be active is determined. A common approach is to compute the Tanimoto coefficient between the fingerprints of each molecule. Kadurin et al.[120] used this method to search PubChem for molecules similar to the fingerprints generated by their model.

4.4.2 *Transfer Learning*

Despite that a simple screening of unspecific model outputs may lead to finding molecules of interest, the process is somewhat inefficient as a large number of the generated molecules end up being discarded. A less wasteful approach would be to first bias the model towards producing (more) molecules meeting the desired properties. This can be achieved with Transfer Learning, a training procedure where a model first learns to perform a similar task, but for which larger datasets exist, being later fine-tuned on the intended data [44]. This approach assumes that several of the underlying attributes learned on the first set are transferable to the second set. In molecule generation, this is usually applied on sequence-based approaches where a large dataset such as Zinc or ChEMBL first helps to learn the syntax of the string representation and then a smaller, targeted, dataset biases the model towards particular attributes, such as a given biological activity, as it is illustrated by Figure 20.

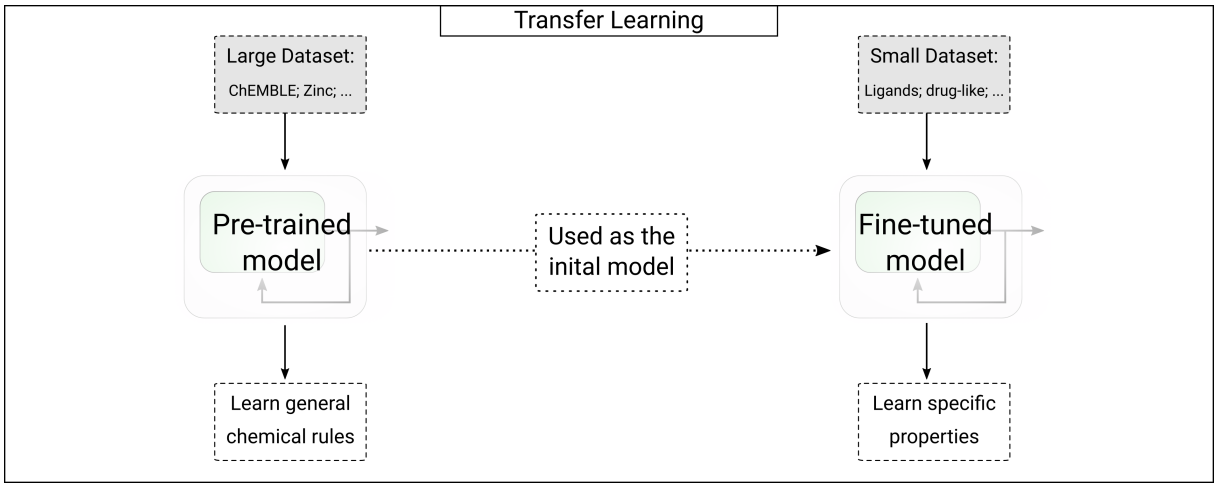


Figure 20: In transfer learning a general model is first trained on a large dataset and then fine tuned towards generating the desired properties with a smaller, focused, dataset.

Transfer learning has been successfully applied to fine-tune stacked RNNs generating SMILES. In 2017, Segler et al.[7] applied this method to a RNN with three LSTM layers generating SMILES. The model was later fine-tuned on known ligands of specific receptors and reported to successfully recover molecules from a hold-out test set.

Merk et al.[8, 9] employed a similar method, where their fine-tuned SMILES-based RNN was used to generate compounds to be later synthesized. The team then performed *in vitro* activity testing, reporting that 4 out of 5, and in a later work 2 out of 4, were active. Gupta et al.[64] experimented with fine-tuning on small datasets, reporting that even just a set of 5 molecules can lead to a model capable of generating unseen actives. Moreover, Moret et al.[89] also looked into the applicability of transfer learning in low data regimes when combined with data augmentation. With just 5 dissimilar natural products, they were able to generate structurally diverse molecules covering a broad range of scaffolds. Departing from sequence-based representations, Gebauer et al.[81] used transfer learning with their point set based model to target a specific value range of HOMO-LUMO gap, a molecular property relevant for the development of organic semiconductors.

Transfer learning can also be used as part of a larger procedure, as a mean to accelerate training and improve results. Notably, Li et al.[92] fine-tuned their model before generating molecules conditioned for a specific bio-activity profile. In a similar vein, Blaschke et al.[90] used transfer learning to focus the model towards features relevant to their objective, facilitating their subsequent reinforcement learning procedure.

4.4.3 *Reinforcement Learning*

A different approach to bias models towards generating molecules of interest is RL. RL outlines a framework where an agent, or system, interacts with an environment through a sequence of actions that are dictated by a policy and evaluated by a reward signal. The agent must iteratively revise the policy to improve the cumulative rewards over the full sequence of actions. This framework aims at learning a system capable of adopting the best set of actions in a given environment [41, 42]. In *de novo* generation of molecules, RL has been applied in both sequence and graph-based approaches.

One application is to first pre-train a model through maximum likelihood estimation and then optimize it in a RL framework towards generating molecules with desired properties. This concept is presented in Figure 21 (top). In this vein, Segler et al.[7] biased their stacked RNN by coupling it to a prediction model and iteratively fine tuning on the generated active compounds. With just 8 iterations, the team reported the successful recovery of active compounds from the test set. Olivecrona et al.[63] trained a stacked RNN to generate molecules as SMILES and, using RL, optimized it towards generating analogs to Celocoxib and generating molecules predicted as active against DRD2. Popova et al.[95] applied this concept to the sequential generation of graphs. Their model, termed MolecularRNN, was first trained to generate diverse realistic samples and then optimized for either Quantitative Estimate of Drug-likeness (QED), melting point, and the logP penalized by SA and large rings (PlogP).

Blaschke et al.[90] developed a production ready generative method, termed REINVENT2.0, based on a stacked RNN leveraging randomized SMILES and reinforcement learning. Later, Blaschke et al.[93] proposed a method to improve diversity in the REINVENT framework, termed memory-assisted RL. This method creates "buckets" grouping similar generated molecules, once a bucket reaches a set capacity, subsequent molecules falling in that cluster are penalized. This memory unit, therefore, helps lead the model to unexplored areas of chemical space. The framework was employed to target a specific range

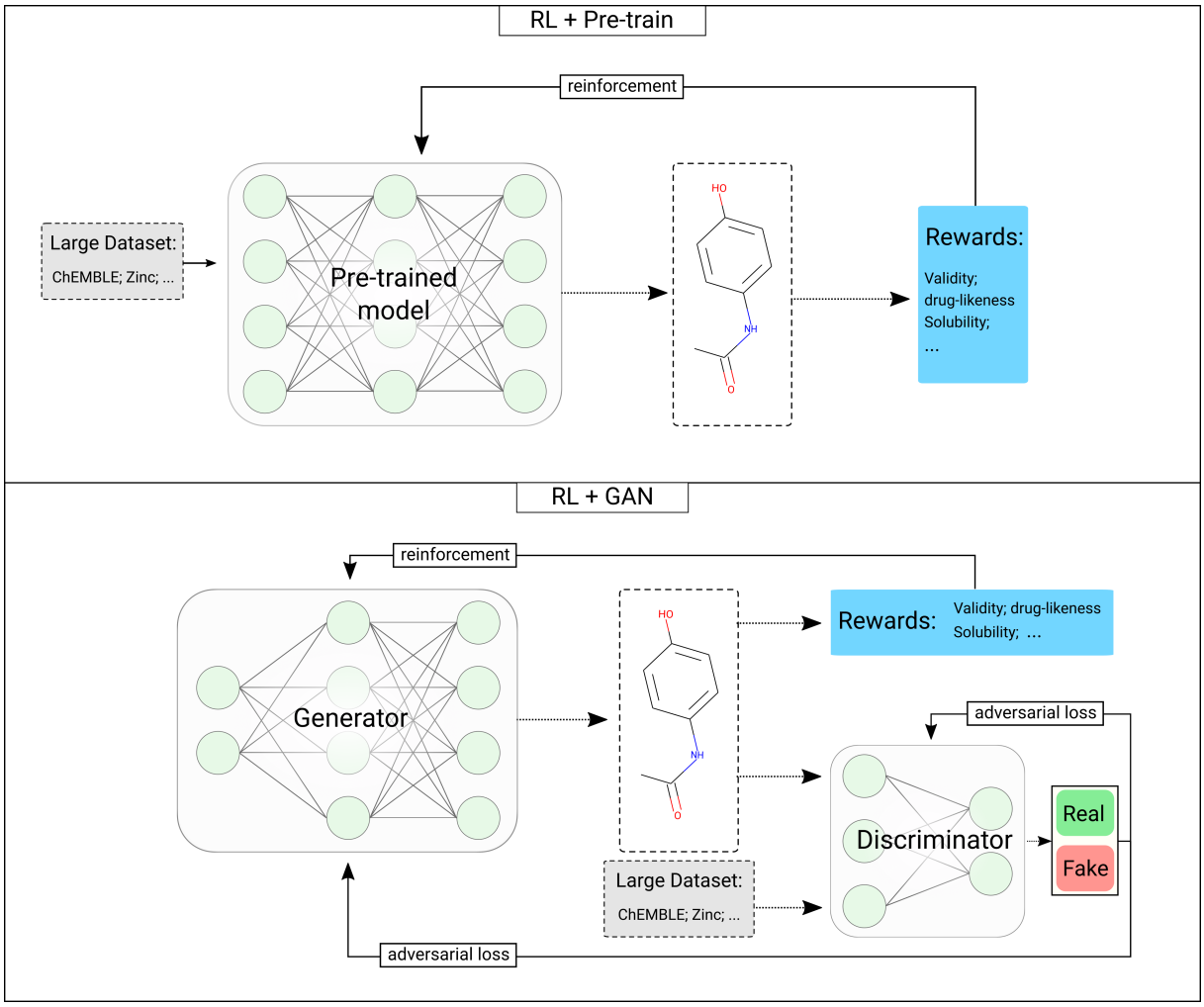


Figure 21: **Top:** The model is first pre-trained through maximum likelihood estimation, learning the structure of the output space along with general chemical rules. Then, using RL the model is optimized for specific properties such as binding affinity or solubility. While similar in concept to transfer learning, the use of RL allows to bias the model towards a wider range of objectives. **Bottom:** Directed generation with RL and GAN. This method leverages adversarial training to produce feasible molecules and RL to bias the generation towards desired properties.

of logP and optimize the predicted activity for HTR1A and for DRD2. The team noted an increase in the generation of diverse scaffolds, while producing highly scored compounds.

Zhavoronkov et al.[94] used RL to optimize a SMILES based VAE towards generating selective DDR1 kinase inhibitors. The objective was based on the predictions of an ensemble of three Self-Organizing Maps (SOM) predicting general activity towards kinases, selectivity for DDR1, and novelty of generated molecules. From the generated compounds, 6 were selected for synthesis and in vitro testing with 2 of those being reported as both active and stable. Further in vivo testing was also performed on 1 molecule, with reasonable pharmacokinetic properties being reported. Lastly, the authors also noted the substantial reduction in both time and costs of their DL based approach compared to traditional drug development pipelines.

GANs and RL can also be combined to generate realistic, but optimized, molecules. Figure 21 (bottom) outlines this method. More specifically, a GAN is trained in a RL framework combining the adversarial

reward with other relevant objectives. This method was employed by Guimaraes et al.[10] in ORGAN to generate SMILES optimizing molecular properties such as logP, Synthetic Accessibility (SA) and QED. Improving on the previous method, Sanchez-Lengeling et al.[96] proposed ORGANIC, optimizing for melting point, drug-likeness with QED and Lipinski's rule-of-five and finally for non-fullerene electron acceptors for use in organic solar cells. Putin et al.[97] proposed ATNC, improving ORGANIC with a differentiable neural computer as generator and a novel reward function to improve the diversity of generated structures. With this model, the team optimized for similarity to known kinase inhibitors and synthesized a molecule similar to a generated one.

With a similar method but departing from sequence-based generation, You et al.[99] proposed GCPN to sequentially generate molecular graphs with optimized properties. Reporting that it can be used to target specific ranges of logP and molecular weight and also optimize the PlogP, while constrained by similarity to a starting molecule.

Karimi et al.[100] employed a similar molecular generative process to generate new drug combinations. Specifically, the proposed method aimed to directly generate sets of novel molecules that could be useful as disease-specific drug combinations. To this end, the team employed a RL process to sequentially generate sets of molecular graphs guided by a chemical validity reward, an adversarial reward enforcing SA and drug-likeness, and a network-based reward to help target the desired disease by incorporating prior knowledge from gene-gene, gene-disease, and disease-disease networks.

Also combining GANs and RL, De Cao and Kipf[76] proposed to generate molecular graphs in a one-shot fashion in their model MolGAN.

4.4.4 *Exploration and Exploitation of Molecules Latent Space*

Instead of optimizing models for the desired properties, models based on the AEs architecture provide a latent representation of molecules that can be used for property optimization or targeted generation of compounds. These methods, which are illustrated in Figure 22, make use of well structured latent spaces, for which VAE and AAE are common choices. Different approaches have been suggested to navigate and shape the latent space of models leveraging both sequence and graph-based representations.

Bayesian optimization

Bayesian Optimization (BO) is a sequential model-based optimization method suitable for black-box problems. It has two main elements, a probabilistic surrogate model and an acquisition function. The surrogate serves to estimate the objective function given some currently known data, then the acquisition function leverages the model to determine the best point in the objective function to evaluate. These new data are then used to update the surrogate model and the process repeats for a set of iterations, ideally leading to the global maximum of the objective function. Common choices are a Gaussian Process for the surrogate and the Expected Improvement for the acquisition function [121, 122].

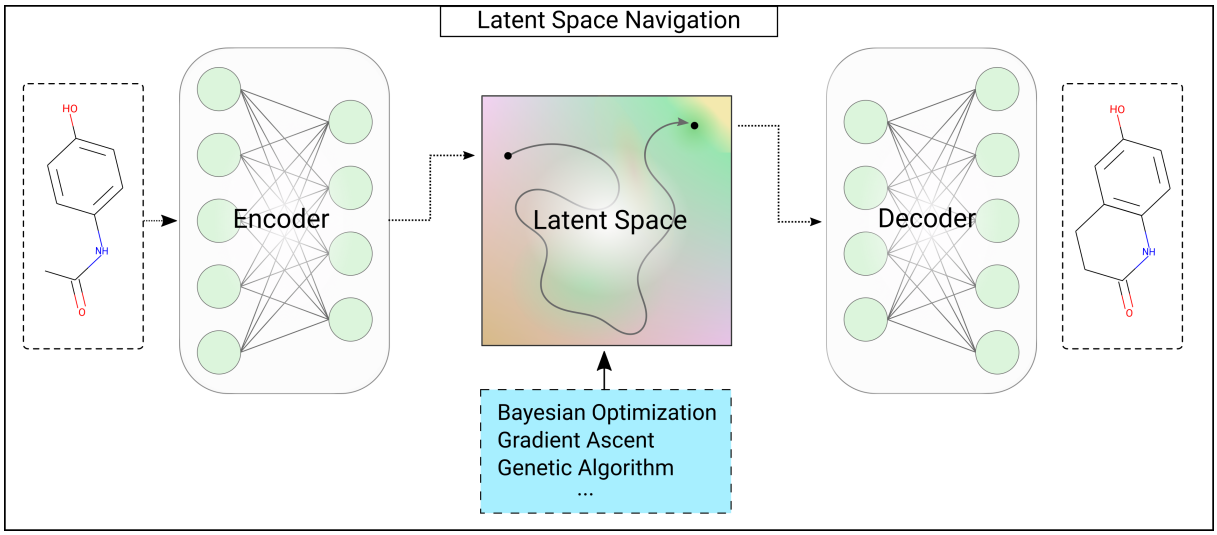


Figure 22: Here, the latent space of an AE is used as a reversible and continuous molecular representation allowing for the application of various optimization algorithms.

In the *de novo* design, BO is used to optimize the properties of molecules by operating on their latent representation, using the decoder to reconstruct molecules from the suggested points. In this application, approximate inference is often used, in the form of a sparse Gaussian Process, due to the large number of evaluations that are made.

BO has been often used to demonstrate that the latent space of a particular architecture can be effectively navigated. In the context of molecular generation, it was first suggested by Gómez-Bombarelli et al.[66] who, in an earlier version of their work, optimized the PlogP. This methodology, and objective function, was then adopted in subsequent approaches by Kusner et al.[72] with GrammarVAE and Dai et al.[102] with SD-VAE which leveraged SMILES production rules and by Jin et al.[103] in JT-VAE and Samanta et al.[104] in NEVAE which dealt with molecular graphs. A more practical objective was employed by Blaschke et al.[11] who optimized the predicted DRD2 activity, reporting that BO was capable of effectively navigating the latent space of their uniform AAE and find novel active molecules. Lastly, Griffiths and Hernández-Lobato et al.[101] applied constrained BO as a way to mitigate training set mismatch, where the BO would visit latent points far from the training data that the model struggles to reconstruct. More specifically, the acquisition function was altered to only consider latent points which decoded to valid molecules with a tangible molecular weight. This was reported to lead to the generation of higher quality molecules using three drug-likeness metrics.

Genetic Algorithms and Particle Swarms

Once a continuous latent space is obtained, other optimization algorithms can be employed to optimize for desired properties. For example, Sattarov et al.[109] used a Genetic Algorithm (GA) to explore the latent space of their SMILES based *seq2seq* AE. Setting as goal optimizing molecules for activity towards the adenosine A2A receptor, they reported the generation of libraries enriched with actives and novel scaffolds. Winter et al.[110] also explored the latent space of a SMILES based *seq2seq* AE using a different meta-

heuristic, Particle Swarm Optimization (PSO). They experimented with optimizing different properties, such as QED, PlogP, activity towards EGFR, and activity towards BACE1. Furthermore, Multi-Objective (MO) optimization was also attempted by minimizing and maximizing activity for each of the receptors, reporting molecules with the desired activity profile and favorable absorption, distribution, metabolism, excretion, and toxicity properties.

Gradient-based Methods

Gradient-based methods are alternative optimization algorithms that additionally require the objective to be differentiable. Although fulfilling the differentiability requirement might not always be possible, training a secondary neural network to predict the desired properties, in parallel with the main model, allows to obtain the gradient of a latent encoding with regards to the so desired property. This process has been used to optimize chemical properties through gradient ascent in some approaches, mainly as a benchmark for the smoothness of latent space.

Jin et al.[103] optimized the PlogP constrained to a set degree of similarity to the starting molecule. More specifically, the team used a feed-forward network as a predictor and constrained the optimization with the Tanimoto similarity to the original molecule. Liu et al.[74] employed a similar methodology, optimizing however for the QED based on the gradients of a gated regression network. Bresson and Laurent[105] used gradient ascent with a single multilayer perceptron to, and following the work of Jin et al.[103], optimize the PlogP and the PlogP constrained by similarity.

Deep Learning for Latent Space Navigation

Deep Learning models can be trained to operate on molecular encodings of a separate generative model. Prykhodko et al.[107] applied such an approach to their LatentGAN, a GAN trained to generate latent points of a separate SMILES hetero-encoder. The real data used for training is obtained by passing SMILES through the encoder of the AE, while during the generation the novel latent points are converted into molecules using the decoder part of the AE. By training their model to produce realist encodings of compounds with activity for either EGFR, HTR1A, and S1PR1 (separate model for each), they reported the generation of valid and novel SMILES with a large percentage predicted as active. In a similar vein, Maziarka et al.[108] proposed mol-cycleGAN, a cycleGAN operating on the latent space of the JT-VAE [103] to optimize molecules, while keeping the results similar to the starting compound. Specifically, the model learns to convert molecules (points in latent space) from one set into another and back again. For example, it can convert from a set with only 3 aromatic rings into a set with only 2, all while keeping transformed points similar to the original. The team reported that the model was capable of removing halogen moieties, replace bio-esters, alter the number of rings, and increase the predicted activity towards the DRD2.

More recently, Chenthamarakshan et al.[111] applied a VAE generating molecules as SMILES and achieved controlled generation with Conditional Latent Attribute Space Sampling (CLaSS) [123]. Under CLaSS, a Gaussian mixture model is trained to match the posterior of the trained encoder and a binary classifier trained for predicting desired properties from latent encodings. Then, samples are drawn from

the mixture model and filtered using the latent classifiers with only those predicted to have the specified attributes being decoded back to sequences. This method was employed to generate possible leads targeting SARS-CoV-2, using latent attribute predictors for three molecular properties and the binding affinity to relevant protein targets.

4.4.5 *Conditioned and Semi-Supervised Generation*

An alternative to both biasing the model and latent space optimization is to include explicit inputs to the model for controlling the properties of the generated molecules. As illustrated in Figure 23 (top) with an AE, this is achieved by introducing a condition vector to the models' input, effectively biasing/conditioning the generation process towards the specified values. During training, the condition vector corresponds to various properties of the encoded molecule, leading the model to infer a correlation between the two. Later, during sampling, this vector can be altered, controlling the properties of the generated molecules.

Li et al.[92] used a sequential graph generator based on GNNs and added a conditioning vector at each step of the graph building process. Their model was conditioned on molecular scaffolds, QED, SA and, after fine-tuning, the predicted activity towards JN3 and GSK-3β. The team was able to successfully generate inhibitors for either receptor, as well as dual inhibitors. A similar process was employed by Li et al.[73] to condition a GNN for sequential graph generation by appending the condition vector solely to the initial node states. Conditioning on the number of atoms, bonds and aromatic rings, the model was able to extrapolate and successfully generate molecules when conditioned with values outside the training data.

Kotsias et al.[112] conditioned a SMILES based stacked RNN by setting the conditioning vector as the initial internal state of the network. Two different approaches were compared, either conditioning with molecular fingerprints or building the condition vector with molecular properties such as Topological Polar Surface Area (TPSA), molecular weight and bio-activity. They reported that while both approaches could generate molecules satisfying the desired properties, the fingerprint-based model generated molecules with scaffolds similar to the seed compound, facilitating the encoding of structural restrictions. Meanwhile, the property-based model generated more dissimilar scaffolds, enabling a more versatile exploration of chemical space.

The conditioned generation methodology is also applicable to AEs, where the condition vector is generally appended to the input of both the encoder and decoder. For example, Simonovsky and Komodakis[77] demonstrated conditioned generation with their GraphVAE by controlling the number of heavy atoms in generated molecules. Lim et al.[67] used a conditioned VAE to control molecular weight, TPSA and number of H^+ donors and acceptors, reporting independent control of properties, as well as generating molecules with properties beyond those seen during training. Working with a 3D representation of molecules, Skalic et al.[80] proposed to condition their shape-based VAE with the location of pharmacophores. Specifically, the decoder received a 3D shape constructed by placing property points close to atoms with that property. The team noted that conditioning the reconstruction of random latent points often led to implausible output

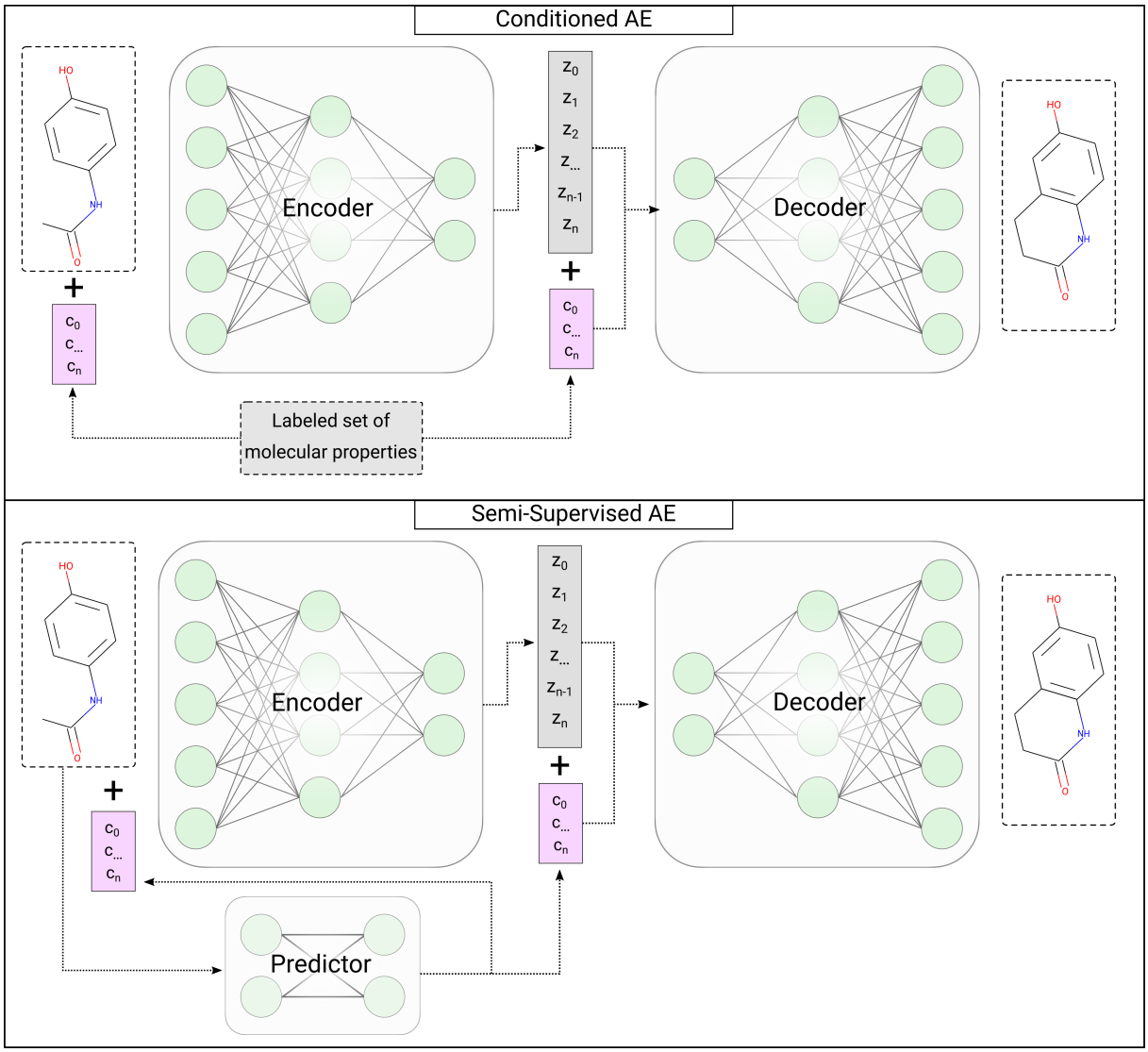


Figure 23: **Top:** In conditioned generation, the desired properties are introduced as explicit inputs to the model. These properties are pre-computed for each compound of the training set and used during training to induce a correlation between the two. This correlation is then leveraged during the generation process to target specific property values. **Bottom:** In the semi-supervised case of conditioned generation, only part of the training set has the desired properties available. To overcome this, a predictor network is trained on the labeled instances and used to predict the properties of unlabeled ones.

shapes, but conditioning the decoding of seed molecules improved the reconstruction of pharmacophore features.

When the desired properties are not readily determinable and no large labeled datasets are available, common with bio-activity data, semi-supervised-AE can be used, Figure 23 (bottom). A semi-supervised-AE consists of an AE with an added predictor network, which receives the molecule as input and outputs its properties. These are then appended to both the input and output of the encoder, ultimately conditioning the decoder. The architecture is termed semi-supervised because the dataset does not need to be fully labeled. Labeled instances are used for training the predictor, replacing its output, while unlabeled samples have their properties predicted by the predictor network [124].

Kang and Cho[118] employed a semi-supervised-VAE conditioned on molecular weight, logP and QED and experimented with various fractions of labeled/unlabeled data used for training. Polykovskiy et al.[69] evaluated the application of different disentanglement techniques to a semi-supervised-AAE. With their most successful method, termed semi-supervised entangled AAE, the team was able to generate molecules conditioned on the activity towards the Janus kinase 2 and Janus kinase 3. By setting low activity for JK2 but high activity for JK3, they generated a set of selective inhibitors that were then filtered. A single molecule was synthesized and reported to have *in vitro* activity and selectivity for the Janus kinase 3.

Lim et al.[119] used a conditioned VAE to extend molecular scaffolds, producing molecules with predetermined scaffolds and desired properties. The model was reported to successfully condition the molecular weight, TPSA and logP. Furthermore, a semi-supervised extension of the model was used to design EGFR inhibitors, reporting a significant enhancement of the inhibition potency.

More recently, Méndez-Lucio et al.[114] used a VAE in conjunction with a GAN conditioned on gene expression data. More specifically, a two-stage GAN is trained to generate latent points of a GrammarVAE [72], with its decoder serving to reconstruct the points generated by the GAN. Here, the conditioning vector is an input to the generator, alongside Gaussian noise. Also, two networks, one per stage, predict whether the generated latent points correspond to the gene expression profiles used for conditioning the generator. Using this method, the team conditioned the model on the gene expression of ten knock-outs of pharmacological interest, reporting the generation of molecules similar to known active compounds.

Born et al.[115] proposed PaccMannRL, a framework that leverages gene expression data and combines reinforcement learning with conditioned VAEs. This method starts by training two separate VAEs, one to reconstruct molecules, represented as SMILES, and the other to reconstruct gene expression data. The two models are then combined, with the output of both encoders being summed together and used as input to the molecular decoder. This new architecture is then trained through reinforcement learning towards generating molecules targeting the specified gene expression profile. The framework was then employed to generate anti-cancer compounds, with the team reporting an improvement in predicted efficacy while maintaining similar validity scores.

Also exploiting omics data, Shayakhmetov et al.[113] proposed a SMILES based conditioned AAE to generate molecules capable of inducing a desired transcriptomic change. Particularly, the model is composed of two AAEs, one tasked with reconstructing molecules and the other with reconstructing gene expression profiles, with the conditioning vector being produced by the gene expression encoder as a separate latent vector. As such, this architecture produces a three-part latent space, with one part meant to encode molecule specific information, another one to encode expression specific information, and the last to encode features relevant to both. This specific arrangement was designed to aid the model in ignoring nonrelevant cellular processes that are included in the gene expression profile from the changes induced by the molecule.

Masuda et al.[117] proposed to condition their previously proposed model, generating molecules as 3D structures, with the 3D structure of the intended binding site. Specifically, a separate encoder module was added to encode the binding target structure into a latent representation, which was then concatenated

to the output of the molecular encoder, conditioning it. With this approach, the team reported that, by sampling around a seed molecule, they often could generate new compounds with better binding affinities.

A different approach was taken by Jin et al.[116] who adapted JT-VAE to optimize input molecules by adding adversarial training and a conditioning vector to the latent encoding. That is, the model was trained to "translate" molecules missing the desired properties into molecules with those qualities. The adversarial objective ensures that the new molecule has the desired properties, while the conditioning vector directs the generation process. With this method, the team reported success when optimizing for the PlogP, QED and the predicted activity towards DRD2 while constraining by the similarity to the initial molecule.

4.4.6 *Synthetic accessibility*

Successfully applying these methods to practical use cases will inevitably depend on synthesizing the novel compounds. However, ensuring the SA of the generated molecules is a major hurdle that often goes unnoticed.

Seeking to help address this, Gao and Coley[125] discussed and compared three approaches capable of guiding generative models towards synthesizable compounds. Specifically, they considered the use of synthesizability scores and retro-synthetic analysis to filter generated molecules, filter the training dataset or to modify the objective function used for targeted generation. The team reported that, despite useful, the first two methods often proved insufficient. Furthermore, modifying the objective improved synthesizability, but at the cost of the main goals.

Horwood and Noutahi[126] directly addressed this issue in their RL based method by iteratively building novel molecules through a series of chemical reactions. With this approach, the team reported the successful optimization of multiple objectives, while maintaining good diversity among generated molecules and also providing a valid synthetic route for every novel molecule.

A similar approach was taken by Gottipati et al.[127], which also applied RL to generate molecules using a sequence of chemical reactions. Like the previous method, the team reported attaining high scoring generated compounds while ensuring SA.

Bradshaw et al.[128] also proposed to generate novel molecules by recursively combining simple building blocks through a series of chemical reactions. The team proposed two variants of their approach, one where RL is employed to perform targeted generation and, departing from the two previous works, a Wasserstein Autoencoder (WAE) based model. The later one, as discussed in the previous sections, should allow other methods of targeted generation to be used.

4.5 Current Applications

There are multiple proposed approaches to not only generate molecules, but also do so in a directed manner, controlling and optimizing for desired properties. The adopted objectives have more often been employed as a benchmark of the proposed methodologies than as goals in themselves. While some

have been of limited real use, like maximizing the PlogP, others such as optimizing for specific bio-activity profiles can have a more direct application in fields like drug discovery. For example, the swiftness of these methods has recently found use in the creation of therapeutic leads for SARS-CoV-2.

4.5.1 *Drug Development*

The large costs associated with drug development has often led to the implementation of various computational tools to assist and accelerate the process. As such, a large focus has been placed on applying deep generative models to various stages of early drug development.

The most commonly suggested application has been to the *de novo* drug design. Indeed, the bulk of methods here described are capable of generating broad molecular libraries, with a large part capable of focused generation. These libraries can then be used for virtual screening or high-throughput screening, hopefully exploring previously unseen regions of the chemical space.

A different suggested application has been for molecule optimization. Although the usefulness of some objectives has been questioned, the ability to, for example, optimize solubility or improve SA, while maintaining high similarity to an original structure can help inform lead optimization. Furthermore, easily and reliably finding close novel analogs to a molecule, or introduce specific modifications like substituting bio-esters, can also be of practical use. Lastly, some attention has also been given to fragment-based drug development, with methods proposed for growing a molecule from a specified fragment[64] or linking two fragments together[129, 130].

Several different biological targets have also deserved attention. Table 3 summarizes these applications. It notes the various instances of experimental validation of *de novo* generated molecules that have already been performed. These are mostly successful *in vitro* activity testing, with two instances of *in vivo* validation. These successful practical realizations encourage further research into this blooming field.

4.5.2 *COVID-19*

With the recent SARS-CoV-2 pandemic, generative DL methods became an attractive option for the *de novo* design of possible therapeutic leads. Indeed, in the span of a few months, several approaches were proposed and a large number of potential compounds were shared.

Bung et al.[91] biased a SMILES based stacked RNN with transfer learning towards generating possible binders of SARS-CoV-2 proteases. Specifically, a set of 1.6 million molecules from ChEMBL were used for pre-training and then around 2500 protease inhibitors were used for fine-tuning the model. Furthermore, reinforcement learning was used to control other molecular properties such as QED, logP, molecular weight and SA. After screening the generated molecules using docking simulations, the team shared 31 potential leads.

Table 3: Experimental validation of molecules generated with generative DL

TARGET	Directed Generation	Activity			REF
		In silico	In vitro	In vivo	
RXR PPAR	Transfer Learning	SPiDER	5 synthesized Reported: 4 active	-	[8]
RXR	Transfer Learning	SpiDER WHALES	4 synthesized Reported: 2 active	-	[9]
JK3 Selective	Reinforcement Learning	Docking	1 synthesized Reported: Active and selective for JK3	-	[69]
Kinase inhibitors	Reinforcement Learning	-	50 purchased (similar) Reported: 7 active	-	[97]
DRD2 5-HT1A 5-HT2A	Transfer Learning	MT-DNN On ECFP4	1+6 analogs synthesized Reported: Active for the 3 receptors	1+6 analogs 1 active and acceptable safety	[131]
VEGFR-2	Train on actives	Docking	5 synthesized Reported: 3 active and non cytotoxic	-	[68]
DDR1	Reinforcement Learning	SOM Pharmacophore	6 synthesized Reported: 2 active and stable	1 tested Half-life 3.5h	[94]
p300/CBP inhibitors	Transfer Learning	Docking	1+26 analogs synthesized Reported: Active, selective and stable	10 analogs tested Good bioavailability efficacy, safety	[132]
LXR agonists	Transfer Learning	-	25 synthesized 3 purchased Reported: 12 active	-	[133]

Shaker et al.[134] used their SMILES-based in-house model, named Rosalind, to generate molecules targeting the SARS-CoV-2 main protease M^{pro}. After applying filters for binding affinity predicted by docking, QED, molecular weight, structural alerts and predicted toxicity the team shared a list of 40 compounds.

Chenthamarakshan et al.[111], as described earlier, employed CLaSS on a SMILES based VAE to generate compounds with favorable binding to three relevant target proteins of SARS-CoV-2. The controlled sampling leveraged property predictors for QED, SA, logP and binding affinity to a specified protein. Filters were then applied for toxicity, retrosynthesis prediction and binding affinity to the relevant target with docking and a set of 3.5K potential leads was shared by the team.

Zhavoronkov et al.[94] used an internal pipeline leveraging 28 different models and various molecular representations to generate molecules targeting the SARS-CoV-2 main protease M^{pro}. Docking was employed to rank the compounds by their affinity and a set of 10 molecules was shared by the team.

Born et al.[135] adapted their previously proposed framework, PaccMannRL, to generate compounds addressing 41 targets of SARS-CoV-2. Specifically, the molecular VAE was adapted to generate SELFIES (instead of SMILES) and the conditioning was performed based on a protein VAE (instead of gene expression data).

4.5.3 *Organic Photovoltaics*

The design of new organic photovoltaics has great potential to help reduce the dependence on fossil fuels by enabling cheaper and more efficient solar energy [136]. Deep generative methods can be employed to quickly design new molecules targeting desired properties such as a specific HOMO-LUMO gap and high Power Conversion Efficiency (PCE). Indeed, some attention has already been devoted to this application.

For instance, Sanchez-Lengeling et al.[96] applied ORGANIC towards generating non-fullerene electron acceptors for use in organic solar panels, reporting an increase in the average predicted PCE. Jørgensen et al.[137] used a GrammarVAE [72] for generating donor-acceptor polymers optimized for a specific range of optical gap and LUMO energy. Griffiths and Hernández-Lobato[101] employed their proposed constrained BO method towards generating molecules optimized for PCE, reporting that the averaged score of the generated molecules lied above the 90th percentile of the training data. Gebauer et al.[81] leveraged transfer learning to fine-tune their model towards generating molecules targeting a specific HOMO-LUMO gap. Similarly, Yuan et al.[138] biased a RNN with transfer learning to generate donor-acceptor oligomers targeting a specific HOMO-LUMO gap.

4.6 Software / Packages

There are currently some software packages implementing relevant tools, as well as some proposed approaches involving the use of DL in *de novo* compound design. These libraries help standardize research, possibly serving as useful baselines and common frame of reference for evaluations. Some examples are:

DeepChem [139] - Deep learning tools for drug discovery, materials science, quantum chemistry and biology implemented as a python package.

MOSES [85] - Python package implementing 5 state-of-the-art models and a number of useful metrics for evaluation.

GuacaMol [84] - Evaluation framework, as a python package, providing a number of standardized benchmarks for *de novo* molecular design.

5

FRAMEWORK DEVELOPMENT

Aiming at a tool for generating new molecules, we devised a framework for facilitating the implementation, training and usage of generative DL models. Furthermore, as we sought not only to generate novel molecules but ones with desirable properties, a second framework leveraging evolutionary computation was developed to interface with the first. These two frameworks represent the bulk of the technological contributions from this work and are detailed bellow.

5.1 Framework for Generative Deep Learning Models - DeepMolGen

The DeepMolGen framework seeks to facilitate the exploration of the various DL-based methodologies being proposed for the *de novo* molecular design. Figure 24 gives an overview of the main working components of the framework.

The need for this tool arose during the initial survey of the field, where it became evident that simply reusing available implementations would often lead to conflicting methodologies. As such, a standard framework was envisioned and ultimately resulted in the development of DeepMolGen. Its main goal was to allow multiple distinct architectures to be implemented under a common interface handling their training, evaluation and usage. By abstracting the specificities of each architecture, molecular representations, metrics, training schemes and methods for targeted generation could then be independently developed and applied in a standardized manner. This also allowed for multiple architectures to be subjected to the same methodologies and their results directly compared. Lastly, it allowed the trained models to be combined or used interchangeably within pipelines targeting specific tasks.

DeepMolGen was thus developed to be easily extensible and to enable the rapid implementation of new architectures, metrics and molecular representations. Specifically, abstract interfaces were developed to represent each of these functional parts and accommodate the broad set of methods already available in the literature.

A separate focus of this framework was its usability. Thus methods standardizing common tasks such as training models and generating molecules were also created as well as the production of automated reports to improve the interpretability of the models being employed.

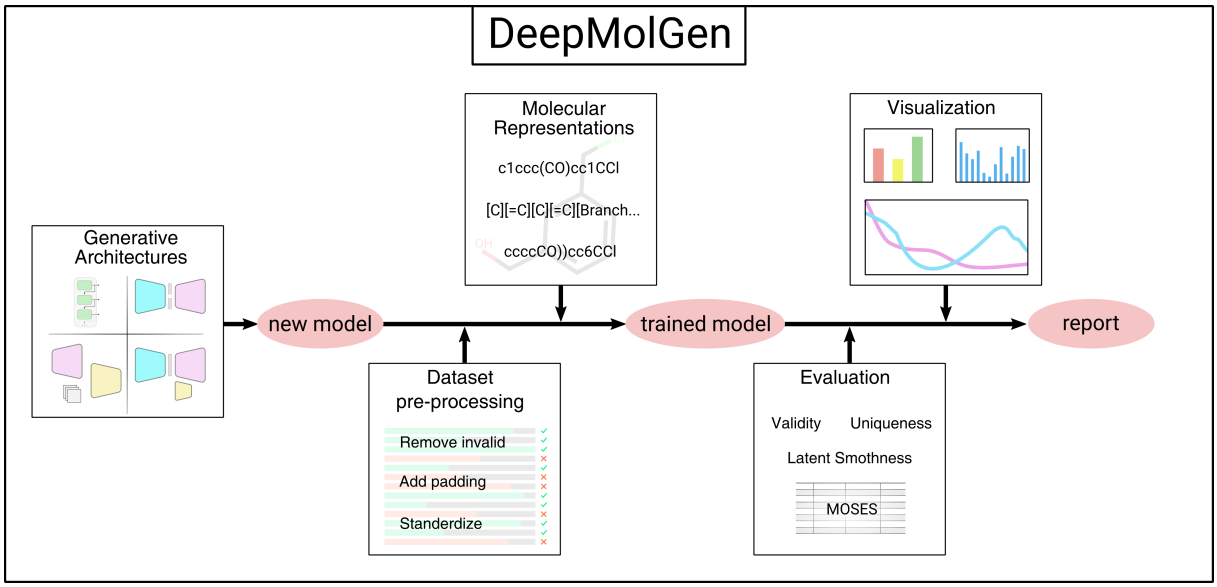


Figure 24: Outline of the DeepMolGen framework, combining multiple generative DL architectures, molecular representations, evaluation metrics and interpretable reports.

5.1.1 *Modules*

The framework is composed of several modules, where some are collections of useful functions, while others are object-oriented, thus defining relevant classes and their methods, and facilitating their extension through inheritance. Each of these modules is detailed below.

File preparation (filePrep): Set of methods to prepare a .smi file, containing molecules encoded as SMILES, to be used in the rest of the modules (Fig. 25(right). Besides standard I/O methods, it can (i) filter invalid SMILES and convert them into the canonical form; (ii) compute the distribution of token lengths, helping to discard outliers which are too long or too short; (iii) pad the SMILES to the same token length; and finally, (iv) join all these operations under a standard and straightforward preprocessing step.

Representations: Classes defining molecular representations to be used by the generative models. Each representation adheres to the same abstract interface, allowing the easy inclusion of new notations and the interchangeability of existing ones. This interface standardizes the translation to and from SMILES, as well as the encoding/decoding of the sequences into representations suitable for the models, such as one-hot encodings. The classes inheritance is presented in Fig. 26. Three string based molecular representations were implemented in the module, namely SMILES[23], SELFIES[71] and DeepSMILES[70]. The first was handled through RDKit and the other two leveraged the packages released with their publications.

Models: Classes defining the DL models, their architectures, training procedures and sampling schemes. As Fig. 27 shows, it has a base class, which every model inherits, defining the structure to follow. This ensures a standardized approach when implementing new models, as most relevant methods can be inherited rather than re-implemented, and guarantees that they are interchangeable with one another and

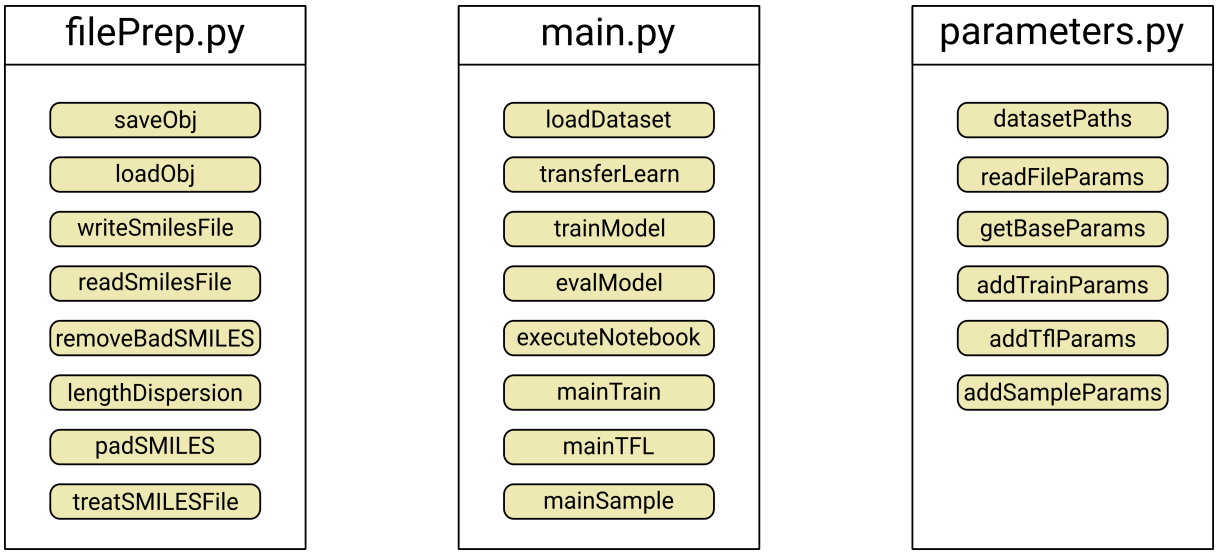


Figure 25: Outline of the '*filePrep*', '*main*' and '*parameters*' modules from the DeepMolGen framework containing various useful methods. The leftmost module handles the I/O and pre-processing of SMILES files. The center module defines the jumping point to the main tasks, with the rightmost module handling their configurations.

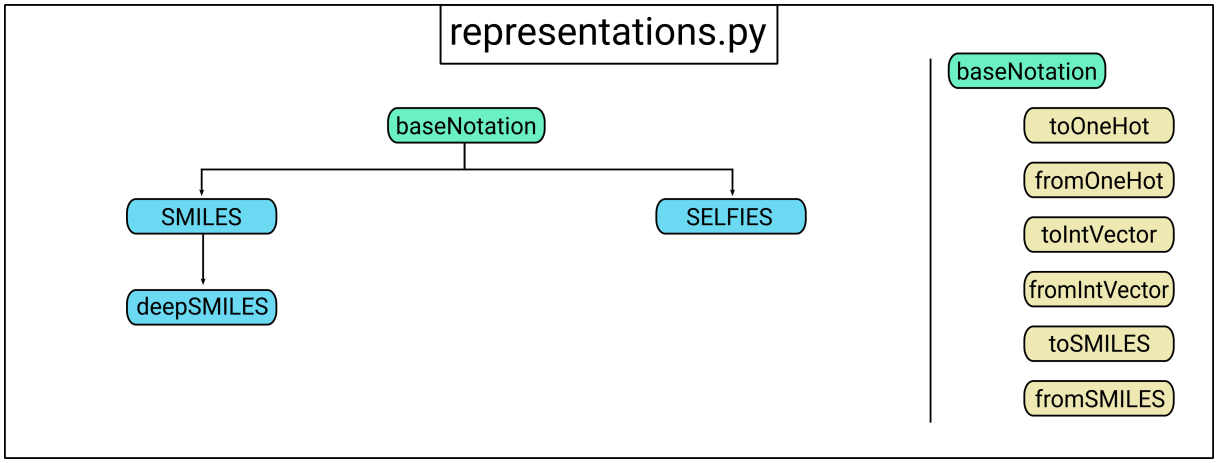


Figure 26: Outline of the '*representations*' module from the DeepMolGen framework. It allows the implementation of new string based representations while ensuring, due to the common interface, their integration with the rest of the framework. The base class is shown in green, its standardized methods in yellow and specific implementations in blue.

compatible with the rest of the framework. Where necessary, base classes for each architecture were also included, with specific implementations inheriting their methods. Specifically, the module implements base architecture classes for AEs, VAEs, AAEs and also simpler stacked RNNs. Then, inheriting from these base classes, 5 state-of-the-art architectures were implemented. These were, respectively, a *seq2seq* based AE 'seq2seqAE', two VAEs 'mosesVAE'[85] and 'blaschkeVAE'[11], one AAE 'mosesAAE'[85] and a stacked RNN[7] 'seglerRNN'. These are further detailed in Chapter 6.1.1.

Metrics: Classes outlining various metrics to evaluate models (Fig. 28). Some are useful for any generative model and widely used in other works, such as the validity, novelty and diversity of generated

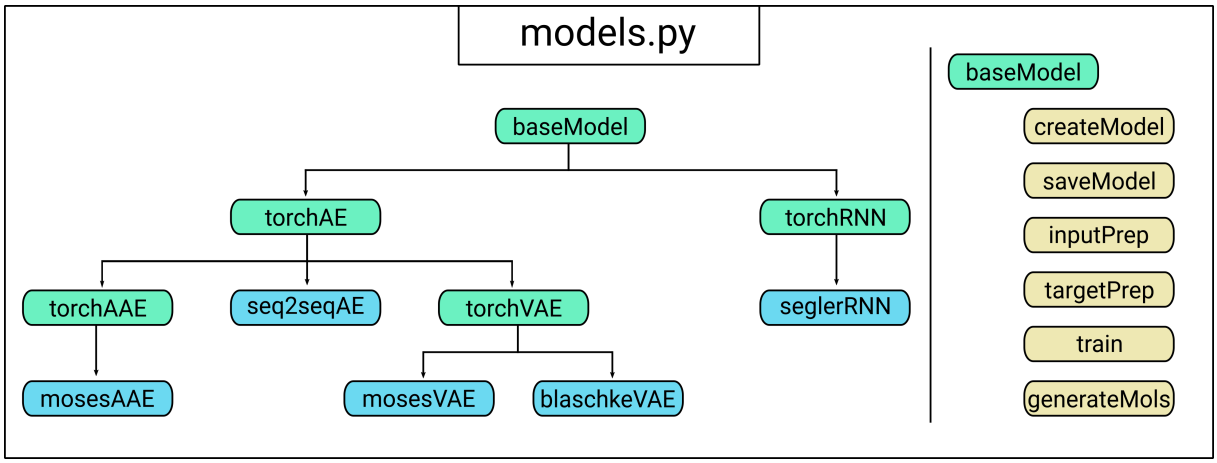


Figure 27: Outline of the '*models*' module from the DeepMolGen framework. The '*baseModel*' class outlines the general interface to follow. Its is then extended with 4 other abstract classes to suit the needs of various architectures. These base classes are shown in green, the standardized methods in yellow and specific models from the literature in blue.

molecules. Others are architecture specific, such as evaluating the smoothness of latent space in auto-encoders as per Blaschke et al.[11]. Others still are designed towards evaluating the progress of transfer learning, like tracking the evolution of the ratio of predicted actives as per Segler et al. [7]. As with the other class based modules, the abstract class enables the easy inclusion of new metrics, as well as their interoperability with the remaining framework.

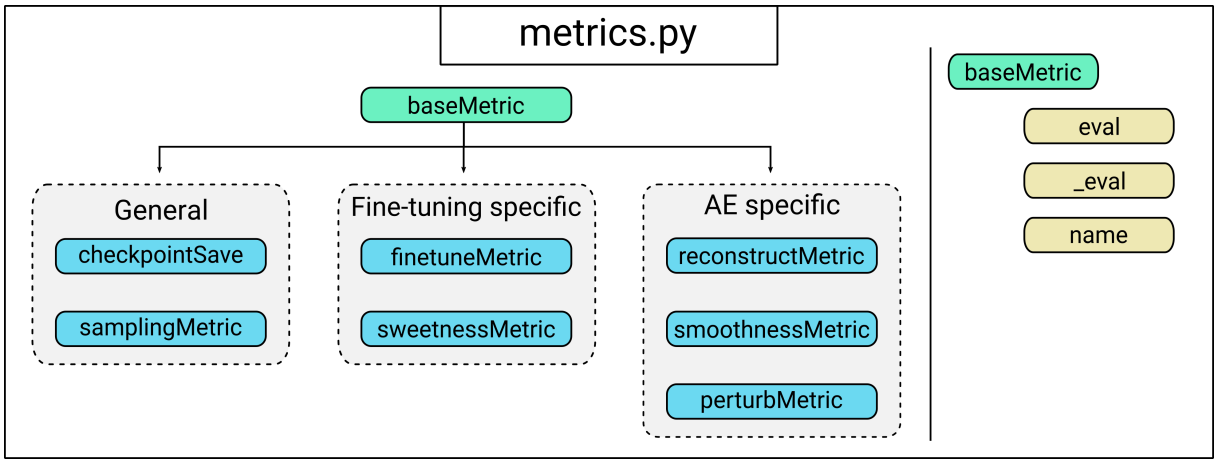


Figure 28: Outline of the '*metrics*' module from the DeepMolGen framework. Besides implementing various useful metrics, this module was also leveraged to create periodic checkpoints during training. The '*baseMetric*' abstract class was a good fit for the implementation of this last task. The base class is shown in green, its standardized methods in yellow and specific metrics in blue.

Main: Methods defining the main tasks that the framework must handle, namely training a new model, fine-tuning a pre-trained model and sampling new molecules. It also handles the data loading and production of the automated reporting. This module, shown in Figure 25 (center), serves to make the overall framework more user friendly by wrapping its inner workings into straightforward methods and also pro-

ducing interpretable reports.

Parameters: Set of methods handling the configurable parameters used in the '*main*' modules tasks. These configurations can be loaded from a JSON file and from the CLI, with the later overriding the first. Some parameters are relevant for all tasks, such as the model and notation to be used, while other are specific for each task, such as the number of epochs or the quantity of molecules to generate. The module also defines some useful default parameters like the path to datasets. Figure 25 (left) shows the methods available in this module.

Descriptors: Module implementing various methods, outlined in Figure 29 (center), producing graphs and figures for use within the automated reporting. The module computes 14 molecular properties of two datasets and either produces various violin plots and barplots for comparing the distribution of properties, or performs a PCA and displays the separation in reduced dimensional space.

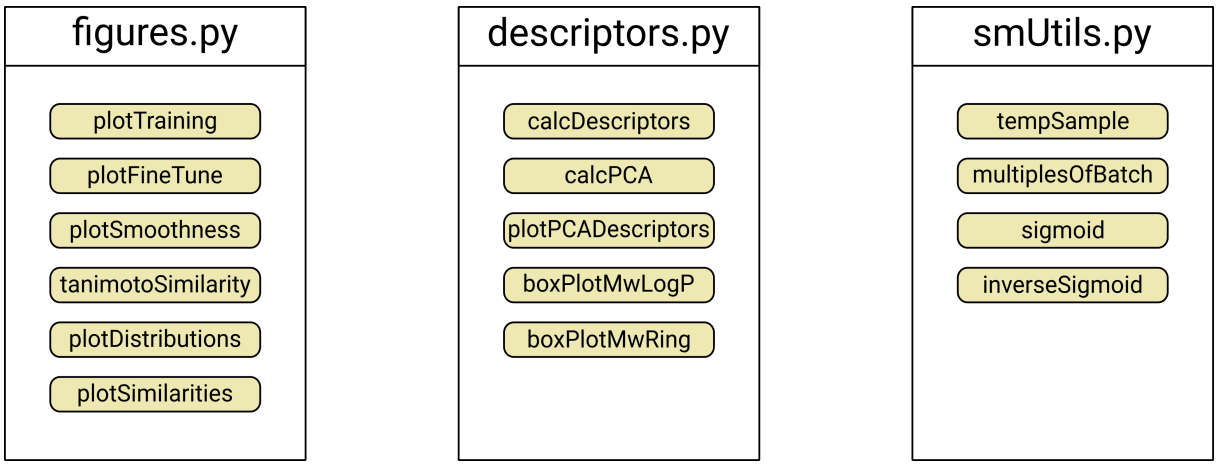


Figure 29: Outline of the '*figures*', '*descriptors*' and '*smUtils*' modules from the DeepMolGen framework. The center and leftmost modules handle the production of illustrations for the automated reporting. The rightmost module collects an assortment of useful methods.

Figures Module containing various methods, illustrated in Figure 29 (left), to produce plots for the automated reporting. The module produces figures with the progress of various loss functions relevant to the models, figures for the metrics relating to latent space smoothness and figures illustrating diversity with the distribution of similarity to the nearest neighbor.

Utilities (smUtils): Set of miscellaneous methods, shown in Figure 29 (left), useful in various stages of the framework, performing tasks such as temperature sampling and the annealing of losses.

5.1.2 *Automated Reporting*

The framework supports automated reporting on training progress and evaluation of models. A blank report in the form of a *Jupyter Notebook* is executed, producing an HTML report with the relevant figures and values (see Fig. 30). Three base reports have been implemented, one for training recurrent models, one for auto-encoder based models and another for reporting on transfer learning. All reports include the configurable parameters used. Both training reports include the evolution of the loss function values relevant to the model, the validity, uniqueness, and novelty, as well as the metrics computed with the MOSES benchmarking framework [85]. The AE specific report then also includes metrics for evaluating the smoothness of the latent space. The transfer learning report similarly starts with the progress of the losses relevant to the model. However, it then plots the evolution of the ratio of predicted actives, the evolution of the distribution of closest neighbor similarities and a series of plots comparing the molecular properties of newly generated molecules against the original training dataset and the fine tuning dataset. These are meant to aid in evaluating the success of the transfer learning process.

5.1.3 *Workflow*

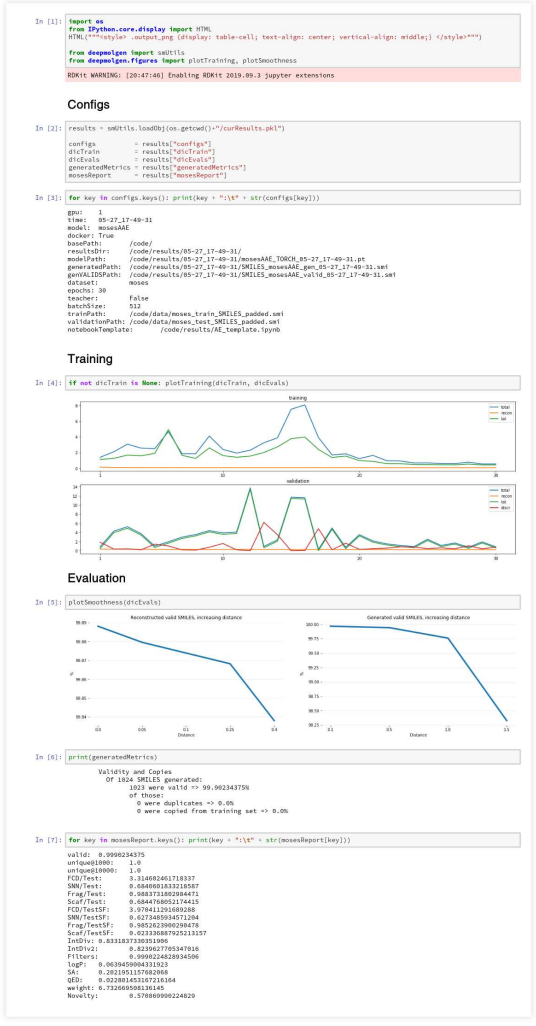
The usage of this framework is performed by calling the main module, passing as arguments the path to a configuration file outlining the task, model, dataset and other relevant parameters. Alternatively, the tasks can be invoked and executed from a user defined script, or other programs may even choose to employ selected methods.

Figure 31 showcases the overall workflow of the implemented tasks. Specifically, training a new model starts by pre-processing a SMILES dataset, defining a set of configurations in a JSON file, and, finally, calling the module with that file. Internally, a new instance of the model, with the appropriate notation, is created and the pre-processed dataset loaded and encoded for the specific model. Training is then performed leveraging the procedures specific to the models class. Once done, the progress of relevant metrics are returned and the appropriate automated report is created.

Fine tuning a model closely follows the previous process, just requiring different parameters. Internally, a pre-trained model is loaded, the dataset prepared, and the model fine tuned. The main differences to the previous process are a smaller default batch size and learning rate and a different set of metrics being computed for the automated report.

Sampling from a model is a simpler task, only requiring the model path, the desired quantity of molecules and an output file to be specified as configurations. Internally, a model is loaded and sampled with the resulting valid molecules being saved to the output file. This process is additionally accompanied with the printing of relevant metrics.

AE_report.html



transfer_learn_report.html

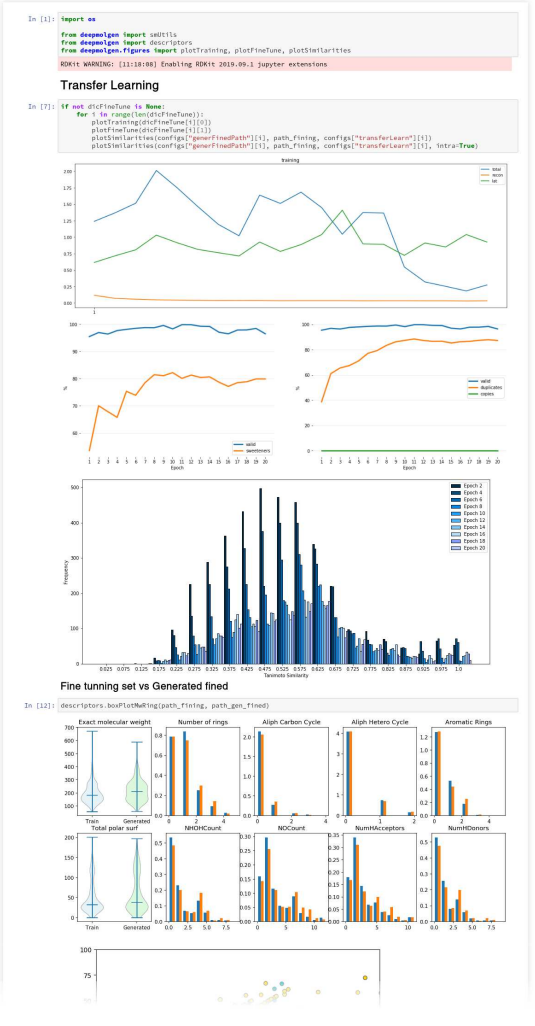


Figure 30: Example of the automated reports produced by the DeepMolGen framework. The '*AE_report.html*' is produced at the end of the training of an AE based model. It outlines the progress of training, evaluation and the relevant metrics. On the left, the '*transfer_learn_report.html*' details the evolution of various metrics but also presents various figures specific to evaluating fine-tuned models.

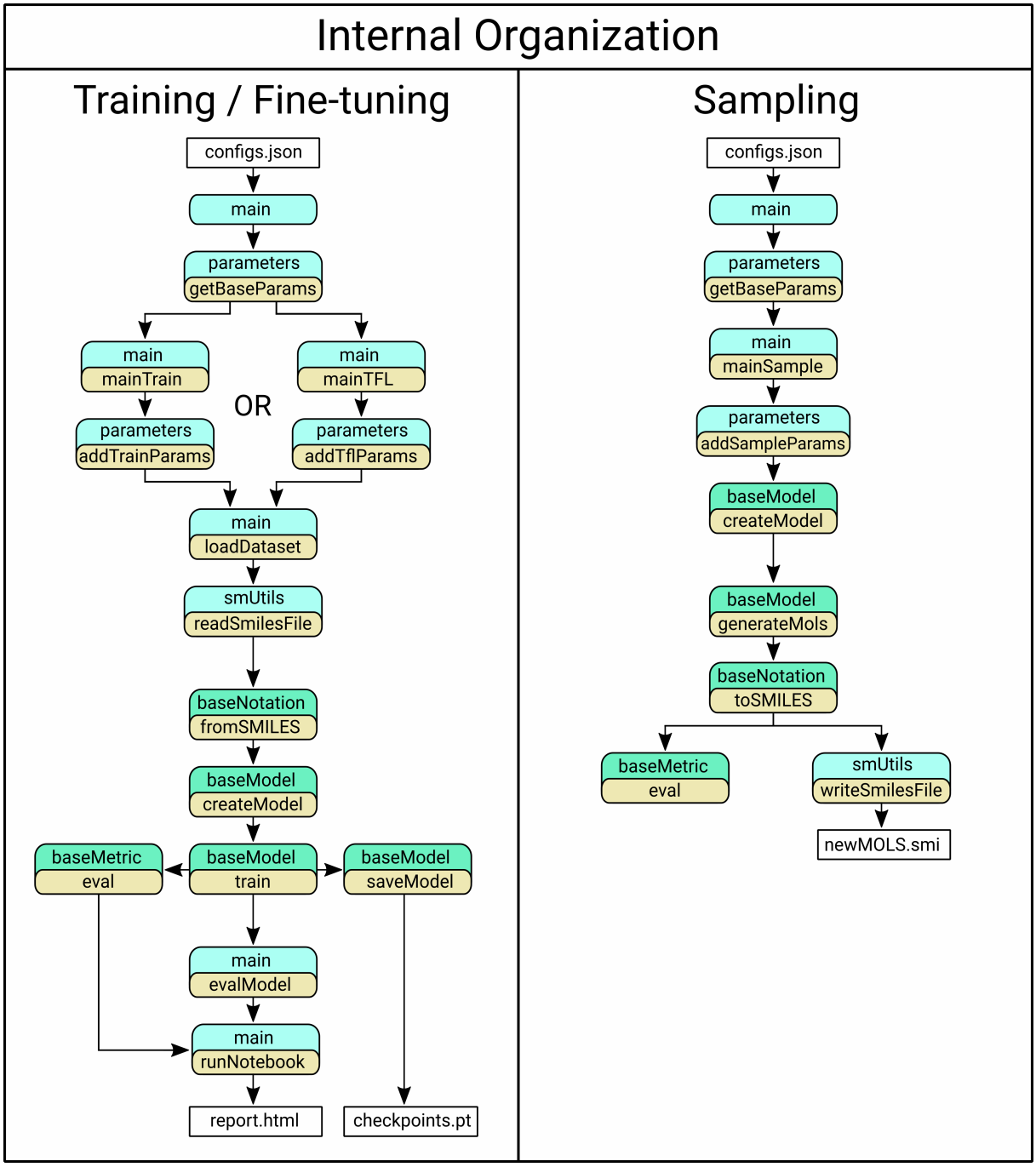


Figure 31: Outline of the internal organization of the three main tasks in the DeepMolGen framework, coordinated within the '*main*' module. Both the training and the fine-tuning tasks are mostly similar. They start by loading the configurations, creating / loading the model and preparing the dataset. This is then followed by training and evaluating the model and finalized with the production of the automated report. The sampling task also starts by loading both the configurations and the model. This model is then used to generate a set of molecules which are evaluated through some simple metrics and saved to a file.

5.2 Framework for Navigating AEs Latent Space with EAs - EAMO

As mentioned in Chapter 4.4, generating novel molecules, while focusing on specific properties is an important avenue of research. The DeepMolGen framework, discussed previously, is already capable of generating targeted sets of molecules through transfer learning. While undoubtedly useful, and employed in this work, this methodology requires a dataset of compounds with the desired properties to bias the model.

Aiming at a more flexible approach, where molecules could be optimized with regards to multiple computed or even predicted properties, we developed EAMO - Evolutionary Algorithms for Molecular Optimization. EAMO takes the latent space navigation approach to targeted generation, as discussed in Chapter 4.4.4, combining an evolutionary algorithm, ML classifiers and a deep generative model. For this last part, the EAMO framework was designed to interface with the DeepMolGen framework, leveraging the architectures and methods implemented within. Figure 32 illustrates the structure of our approach with this framework.

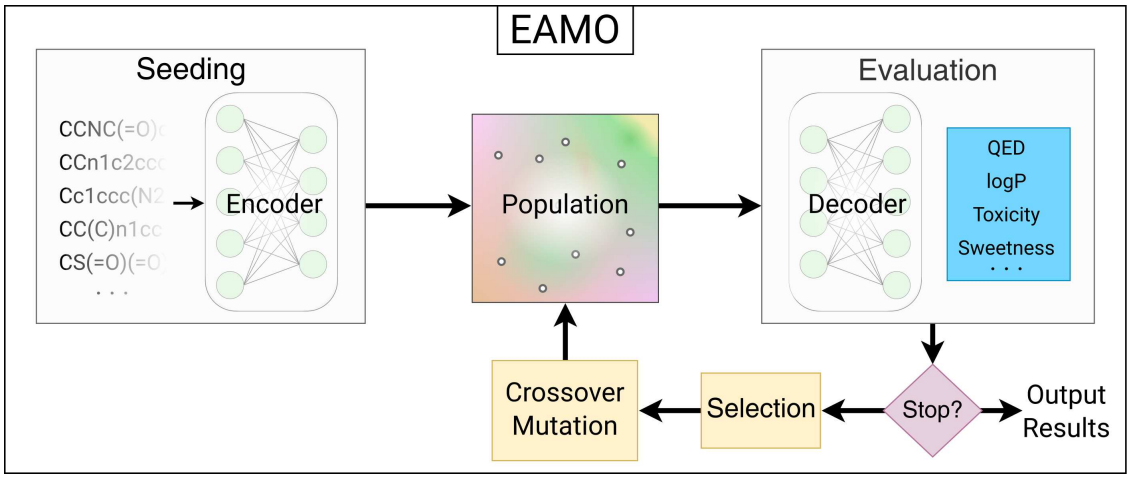


Figure 32: Outline of the EAMO framework, combining generative deep learning and EAs to design molecules.

Specifically, the EA considers each individual, a vector of real numbers, to be a point in the AEs latent space. The encoder part of the molecular AE is leveraged to create an initial population, while the decoder part is used to reconstruct individuals into molecules. These reconstructed molecules can then be scored based on the targeted chemical properties computed through either specific software applications or libraries (such as RDKit) or supervised classifiers. Based on these scores, the EA applies its operators to create a new population and the cycle repeats until a stopping criterion is met.

Our approach uses a standard one-point crossover and two mutation operators:

- Gaussian mutation, which consists of adding a random value from a standard Normal distribution to elements of an individual's vector. Each element has a 10% probability of being mutated;
- Random mutation: one randomly selected element of the vector representation is substitute by a random real value within the range [-10,10].

The initial population is encoded from a supplied set of molecules. If this set is larger than the defined population size, molecules are selected by randomly sampling the provided set. On the other hand, if the provided set is smaller that the intended population, new individuals are created by randomly shuffling the genome of existing individuals.

The proposed architecture allows for both single and multi-objective optimization. While Single-Objective (SO) optimizations are assured by a GA, the MO optimizations resort to the Non-dominated Sorting Genetic Algorithm III (NSGA-III)[140]. Additionally, EAMO also implements two other MO algorithms, namely the Strength Pareto Evolutionary Algorithm (SPEA2)[141] and Generalized Differential Evolution 3 (GDE3)[142].

5.2.1 *Modules*

Similar to the implementation of DeepMolGen, the EAMO framework was developed as several modules, some object-oriented and other collections of useful functions. The framework is meant to be agnostic to the packages and algorithms responsible for the evolutionary computation itself. It currently wraps two state-of-the-art packages, inspyred [143] and jmetalpy [144], that handle those algorithms. Moreover, thanks to its modular design, the inclusion of new packages in EAMO should be a straightforward task. Besides wrapping the evolutionary computation packages, classes and methods were introduced to help interface with DeepMolGen, handle and evaluate molecules as well as standardize the usage of the framework. We now outline the modules that compose EAMO, with a focus on the adaptations introduced specifically for performing molecular design.

Evaluation: Module implementing classes for scoring molecules based on specific properties (Fig. 33). It includes various molecular properties commonly used in the literature [10, 66, 99, 103], leveraging the RDKit package for their computation. These are:

- Molecular weight;
- logP, measuring the molecules solubility [145];
- QED, capturing molecular quality or 'beauty' into a single score [146];
- SA, measuring how easily a molecule can be synthesized which greatly effects its potential as a lead [147];
- Natural product likeness, evaluating the potential for biological interactions as natural products have evolved to perform this task [148];
- Ring sizes, penalizing molecules with very large or very small rings which can be unstable and difficult to synthesize [66];
- Tanimoto similarity between its fingerprint and that of a preset molecule, allowing the optimization of specific structures.

Meanwhile, other scores leverage ML classifiers to predict some intended activity. Specifically, activity towards DRD2, which as been targeted in several studies [11, 63, 108, 112, 116, 131], can be predicted using the SVM introduced by Olivecrona et al. [63]. Likewise, non-toxicity can be predicted using a multi-target deep neural network available from previous work by the research group. Lastly, molecules can also be scored for sweetness using an ensemble combining a SVM a RF and a DNN, also available from the research group.

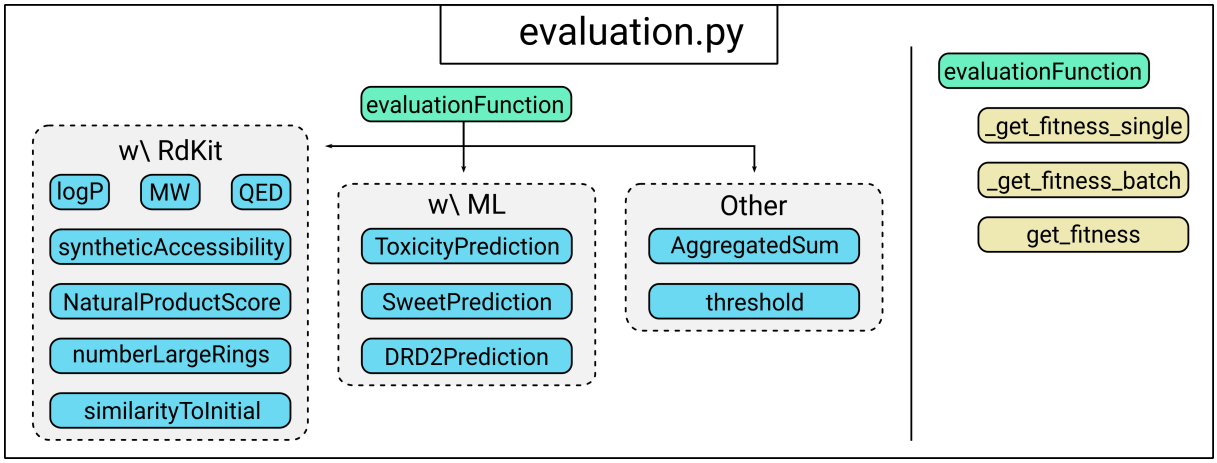


Figure 33: Outline of the '*evaluation*' module of the EAMO framework. The base class is shown in green, its standardized methods in yellow and specific scores in blue. For efficiency, and working with '*problem*', each class can also evaluate molecules in a batch manner. Here illustrated under 'other', the '*aggregatedSum*' class is used to combine multiple other classes of this module under a single weighted average score.

loadModels: Module defining methods to load all the ML models used for evaluation, as well as for interfacing with DeepMolGen to load the deep generative models, Figure 34 (Left). Specifically, it provides access to any model implemented in DeepMolGen which inherits from the '*torchAE*' abstract class, and as such as a latent space.

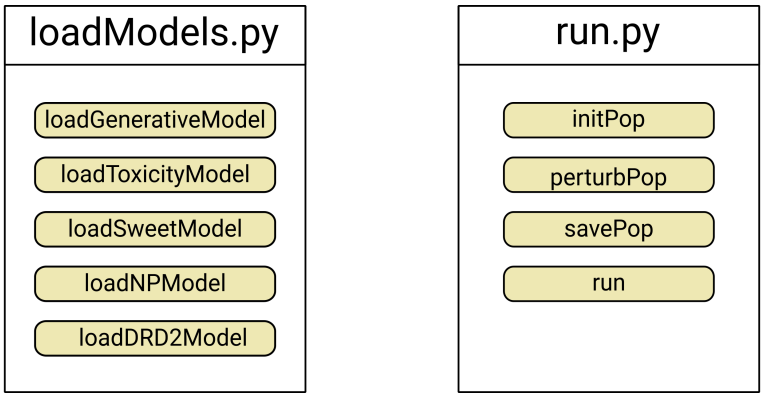


Figure 34: Outline of the '*loadModels*' and '*run*' modules of the EAMO framework. The '*run*' module is the launching point for single optimization runs. To avoid duplicate implementations, it leverages the objectives implemented within the classes of '*caseStudies*' for its '*chemicalProblem*' instances.

EA: Module implementing an abstract class representing an EA, meant to be subclassed for each evolutionary computation engine. As Figure 35 shows, classes for the inspyred [143] and the jmetalpy [144]

packages have been implemented. These map their internal structures and methods, such as selecting the EA to use, setting an initial population or defining the mutation and crossover operators, into a common interface which is then used by the rest of EAMO. This structure allows for easily swapping the engine being used, and also, should the need arise, to include new evolutionary computation packages. Both the inspyred and jmetalpy engines support a wide gamut of EAs for performing SO or MO optimizations. However, the jmetalpy engine also supports several Many-Objective EAs such as NSGA-III. As several of our case studies employed 3 or more objectives, and thus benefited from these Many-Objective EAs, we exclusively used the jmetalpy engine

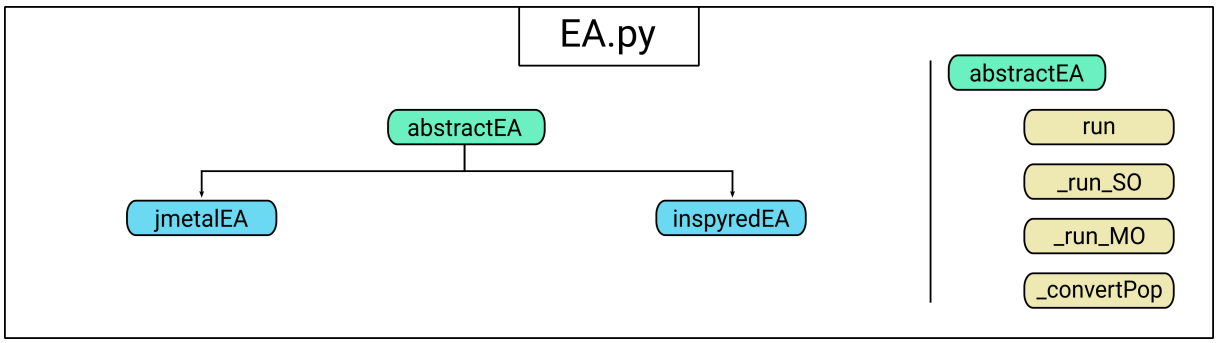


Figure 35: Outline of the '*EA*' module of the EAMO framework. A subclass was created to wrap each engine, providing an interface between our framework and the internal workings of that engine. The base class is shown in green, its standardized methods in yellow and specific engine wrappers in blue.

Problem: Module implementing classes to represent the objective to be optimized, with each engine interacting with this module when evaluating its individuals. As Figure 36 shows, this module was extended with a '*chemicalProblem*' class which, alongside the '*Evaluation*' module, handles most of the 'molecular' aspects of EAMO. Specifically, it interfaces with DeepMolGen to reconstruct the individuals into molecules and scores each one with the appropriate classes of the '*Evaluation*' module. The '*chemicalReporter*' class is a peculiarity of EAMO, providing the exact final scores of the molecules in a population. It is needed to overcome the stochastic process of reconstructing molecules, where a given latent point would decode to several slightly different molecules and lead to a mismatch between the reported scores and the associated molecules.

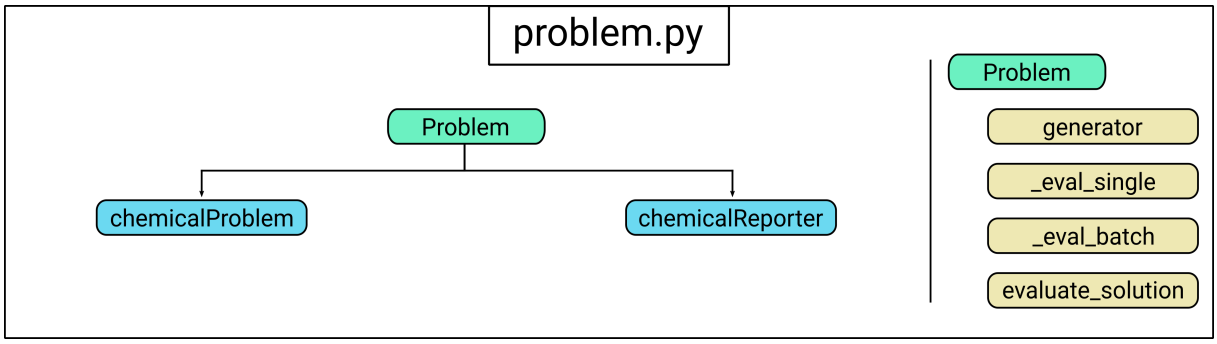


Figure 36: Outline of the '*problem*' module of the EAMO framework. It was extended with a method for the batched evaluation of molecules as that was found to be significantly more efficient.

Run: Module for controlling the optimization run, defining methods for creating the initial population, executing the EA, saving the final solutions found and also coordinating the previous steps under a single method. To perform these tasks this module interacts with the common interface defined by the '*EA*' module, making it agnostic to the engine and algorithm being used. Figure 34 (right) shows the methods included in this module.

caseStudies: Module outlining the procedure for each case study. As Figure 37 illustrates, each case study was implemented as a class, inheriting from an abstract class. It handles the creation of directories, repetition of optimization runs for statistical stability, the post-processing of the resulting data and saving all the results. Moreover, it also defines the optimization objective for each experiment through a '*chemicalProblem*' instance, often combining multiple individual scores from the '*Evaluation*' module. Six different case studies were implemented for the present work, namely the optimization of:

- QED;
- Penalized solubility PlogP;
- Similarity constrained PlogP;
- Activity for DRD2;
- Potential sweeteners.

These experiments will be further detailed in the following chapters. It is worth nothing however, the '*directSample*' class, simply samples the generative model and doesn't perform any optimization. It was included with this module for a more streamlined methodology in our last case study.

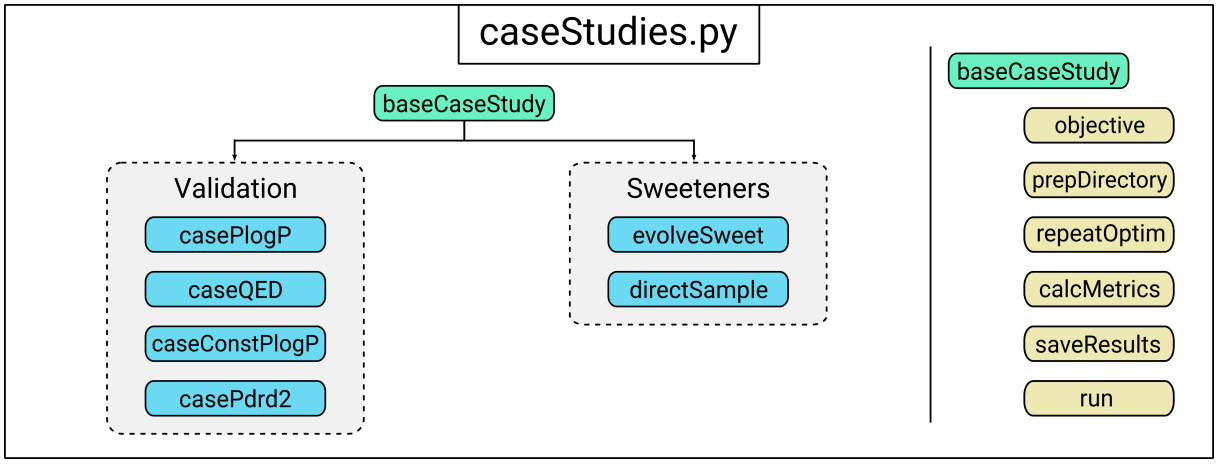


Figure 37: Outline of the '*caseStudies*' module of the EAMO framework. This module is the jumping point to the case studies discussed in this work. Each class implements specific objectives to optimize and oftentimes specific procedures for analyzing the resulting data. The '*run*' method of the base class coordinates the execution of the rest of the methods.

5.2.2 *Workflow*

Similar to DeepMolGen, EAMO relies on a configuration file to define all relevant parameters, such as the number of generations, population size or the DL model. There are two jumping points, either through the '*run*' module or the '*caseStudies*' module.

The first is suitable for single optimization runs, employing one of the objectives defined within the module. Internally, it first creates an initial population by encoding a set of molecules, which are defined in the configuration file. Then, it creates an instance of the '*chemicalProblem*' class to serve as the optimization objective. Both of these are then used to launch the EA and, once evolved, the final population is saved.

Running a case study is accomplished by calling the '*caseStudies*' module. As Figure 38 shows, internally it prepares the directory structure, performs multiple optimization runs through consecutive calls to the '*run*' module, reads and compiles the results of each run into a simple report, and, finally, saves the results to the previously prepared folders.

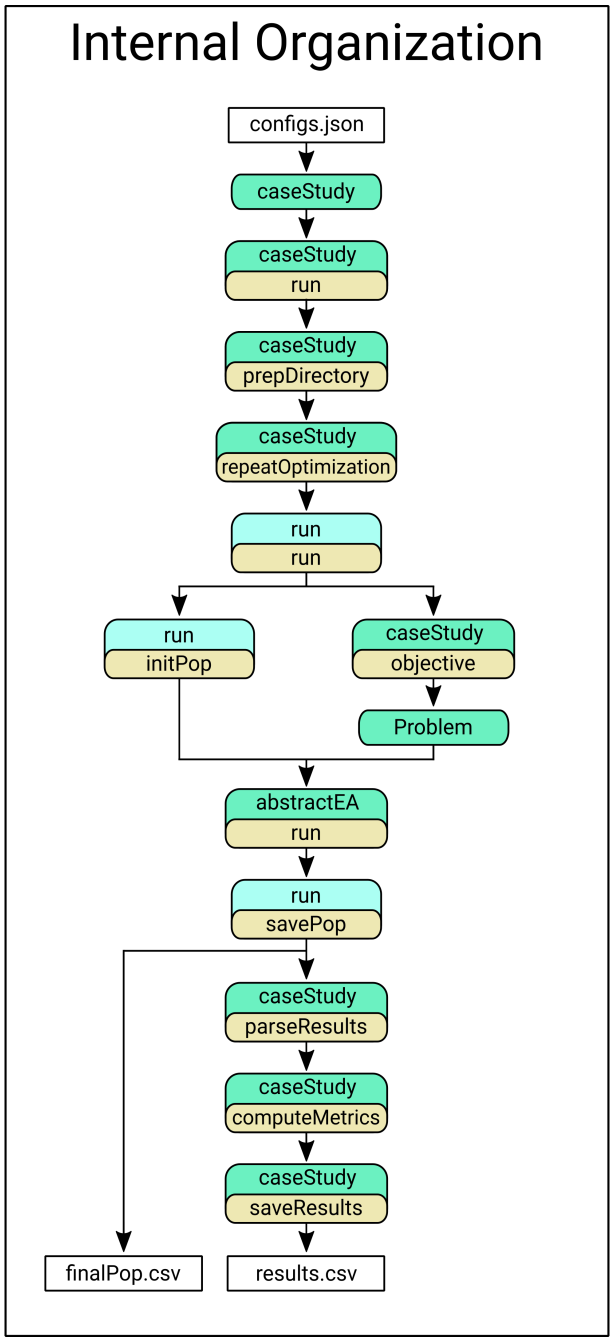


Figure 38: Outline of the internal organization of the EAMO framework. Execution starts by loading a configuration file and instantiating the appropriate '*caseStudy*' class. A folder is then created, followed by multiple optimization runs through repeated calls to the '*run*' method of the '*run*' module. The results are then analyzed and saved, along with any intermediary files.

6

RESULTS AND DISCUSSION

Having developed the two previously discussed frameworks, we sought to validate them through a series of case studies. We started by implementing, training and testing several state-of-the-art architectures within the DeepMolGen framework. Then, using one of these models, the EAMO framework was validated with a set of tasks from the literature. Lastly, we applied these tools to a more practical use case by attempting to generate novel sweeteners.

6.1 Architectures, Training and Evaluation

The first step in validating our methods was to leverage the DeepMolGen framework to implement several architectures. These were then trained with a large dataset taken from the literature and validated with some common metrics as well as with the MOSES [85] benchmark.

6.1.1 *Architectures*

As illustrated in Figure 27, each architecture was implemented as classes within the *'models'* module, inheriting from the closest base class. This allowed for not only a more straightforward implementation but also for better consistency in the methods being applied. For instance, the method for generating new molecules is implemented as part of the "torchAE" class and then inherited to all other AEs. In this work we implemented 1 RNN base model and 4 AE based models, specifically a *seq2seq* based AE, two VAEs and an AAE. However, the RNN based model will not be discussed further as it was implemented as a mere prof-of-concept due to it not being applicable with EAMO. The AE based architectures are detailed next.

mosesVAE: The VAE proposed by Polykovskiy et al.[85], illustrated in Figure 27 as '*mosesVAE*', employs a recurrent encoder and decoder. The architecture is outlined in Figure 39. The encoder is composed of an embedding layer followed by a one layer bi-directional RNN with 256 GRU cells and then two parallel dense layers with 128 units each. The outputs of these dense layers are interpreted as the set of means and standard deviations of a multidimensional distribution. The decoder starts with an embedding layer and a dense layer in parallel. The first, receives the target sequence delayed by one step during training

and, during generation, the sequence being built or the special start token when initializing. Concurrently, the dense layer with 512 units receives the latent encoding vector. A three layer RNN with 512 GRU units in each layer then follows. It takes as input the output of the preceding embedding layer. Meanwhile, the output of the preceding dense layer is used to initialize its hidden state. This is then followed by a dense layer whose number of units depend on the size of the vocabulary of the notation used, i.e. 39 units for the SMILES notation.

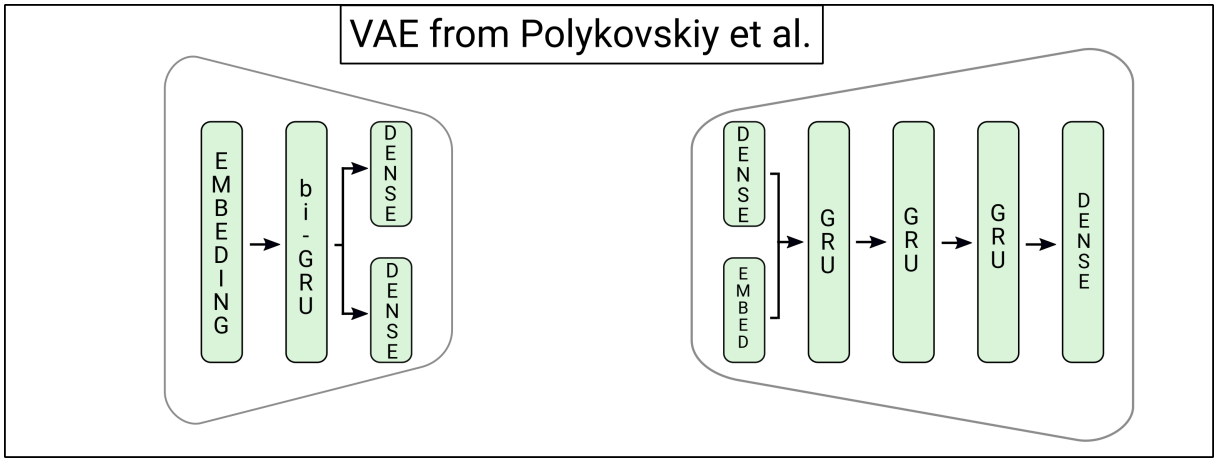


Figure 39: Outline of the VAE based on Polykovskiy et al.[85]. Both the encoder (left) and decoder (right) are RNNs employing GRU cells. Not pictured here, during training the outputs of the two last layers of the encoder are used to sample a single point to feed into the decoder. During evaluation, for instance when reconstructing a molecule, the output previously interpreted as the means is directly feed to the decoder.

blaschkeVAE: The second VAE implemented followed the architecture proposed by Blaschke et al.[11] and is depicted as '*blaschkeVAE*' in Figure 27. The architecture is portrayed in Figure 40 and consists of a convolutional encoder and a recurrent decoder. Specifically, the encoder has three 1D convolutional layers with filter and kernel sizes of 9, 9, 9 and 9, 10, 11 respectively. These are then followed by a dense layer with 128 units and then two more parallel dense layers with 128 units each. Each layer in the encoder, save for the last two layers in parallel, is also followed by a batch normalization layer. The decoder starts with a dense layer with 128 units followed by dropout, p=0.08, and batch normalization layers. Then, a three layer RNN follows with 488 GRU units in the first two layers with the size of the third depending on the size of the vocabulary. A dense layer with the number of units also depending on the vocabulary then finalizes the decoder.

mosesAAE: The AAE implemented also followed an architecture proposed by Polykovskiy et al.[85], presented as '*mosesAAE*' in Figure 27. The general layout of the architecture is outlined in Figure 41 with the auto-encoder part of the network closely resembling that of the VAE by the same team, with three key differences. First, the output of the first dense layer of the decoder is used to initialize the cell state, with the hidden states being initialized to zeros. Second, instead of GRU units it employs 512 LSTM cells per RNN layer in both the encoder and decoder. Lastly, AAEs require a discriminator network which, in this

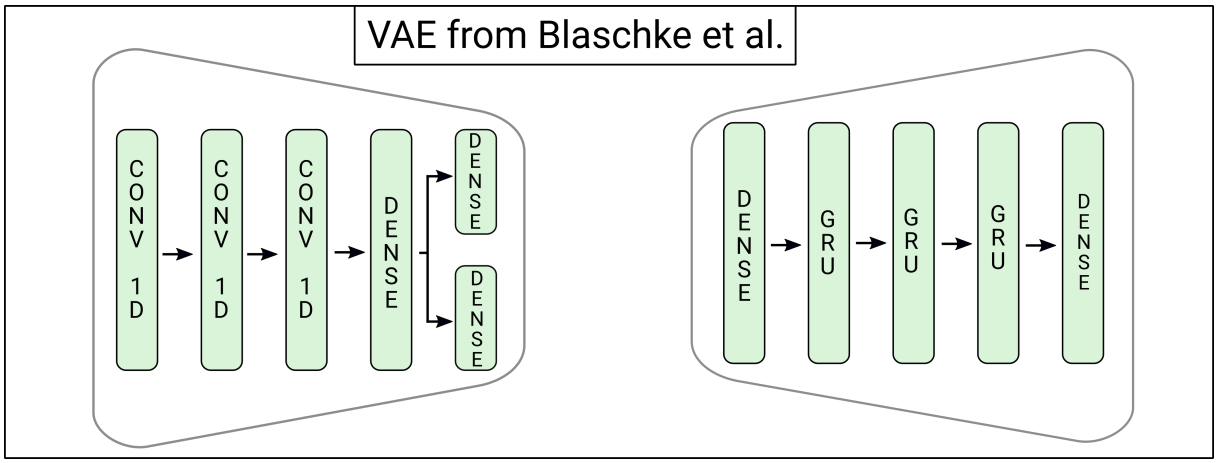


Figure 40: Outline of the VAE based on Blaschke et al.[11] with a convolutional encoder (left) and a recurrent decoder (right). It behaves similarly to the 'mosesVAE' with sampling occurring during training and the means being used during evaluation.

model, is composed of three consecutive dense layers with 640, 256 and 1 units, respectively.

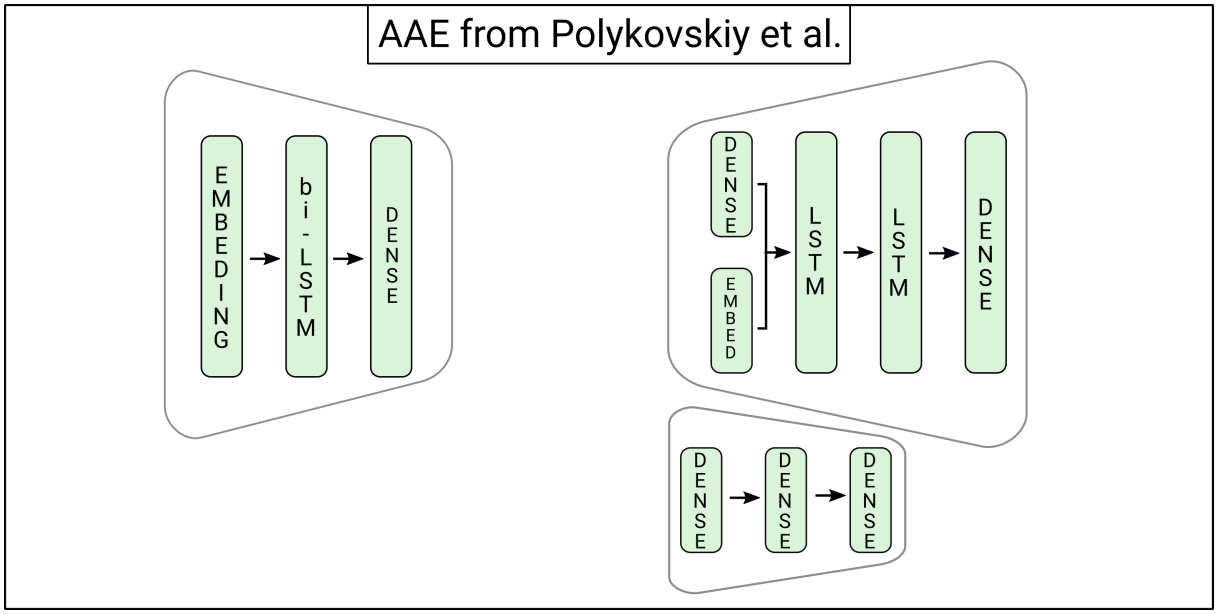


Figure 41: Outline of the AAE based on Polykovskiy et al.[85] with both encoder (left) and decoder (top-right) leveraging LSTM cells in their RNNs. The discriminator (bottom-right) is used during training to regularize the latent space and discarded for the evaluation.

seq2seqAE: Outlined in Figure 42, this is the less complex of the implemented AEs and also employs a recurrent encoder and decoder. The first consists of a single layer RNN with 64 LSTM units followed by a dense layer with 128 units. The decoder part of the network starts with two dense layers in parallel, each with 64 units, and whose output serves to initialize the hidden and cell states of the single layer RNN, with 64 LSTM cells, that follows. It is then finalized with a dense layer with the number of units also depending on the vocabulary used. This architecture borrows from the *seq2seq* architectures in that it uses the final

hidden state of the encoder to initialize the hidden state of the decoder, a characteristic of *seq2seq* models [149, 150]. In Figure 27, this architecture is depicted as '*seq2seqAE*'.

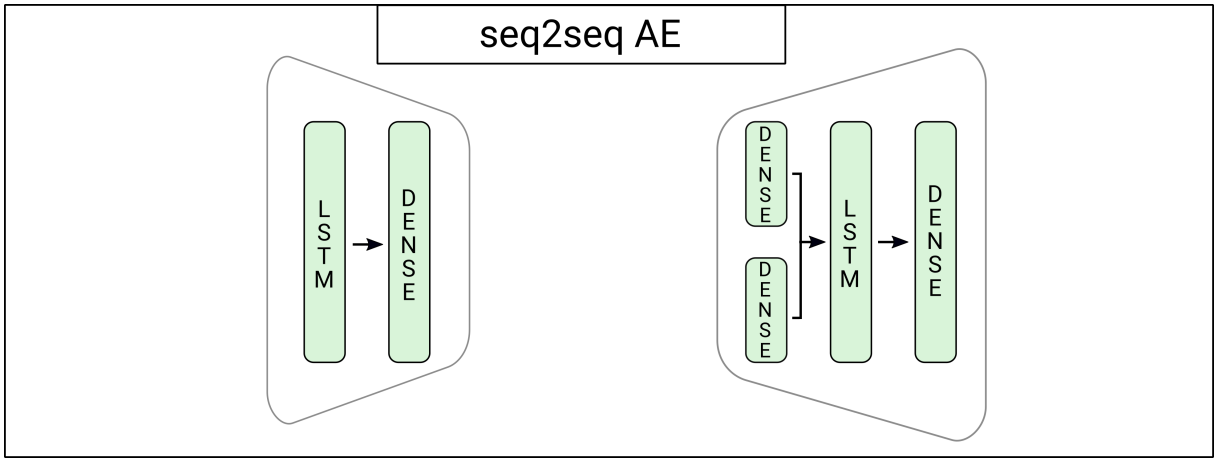


Figure 42: Outline of the AE based on the *seq2seq* arquitectures. Both encoder and decoder employ single layer RNNs with LSTM cells.

6.1.2 *Training and Evaluation*

The models were trained with the dataset provided by [85], a subset of the ZINC[29] database with 1.7M molecules divided into 1.5M for training and 170K for testing. The ADAM optimizer was employed with the default parameters and a learning rate of 0.001. Furthermore, a batch size of 512 was used as well as gradient norm clipping. Lastly, each architecture was trained until no significant improvements were observed, with checkpoints being created every 10 epochs.

For the VAEs, the loss term restricting the latent encodings to be close to the standard normal distribution was annealed, that is, its proportion in the total loss was slowly increased during training. In practice, this was implemented by weighing the loss term depending on the current epoch with the weight being determined by a modified sigmoid function:

$$y = \frac{1}{2 + e^{-0.35 * x + 4}} \tag{1}$$

where x is the current epoch. This function was tailored for our specific use case of around 40 epochs of training, gradually increasing over the first 15 epochs and then plateauing onwards with a value of 0.5. A useful improvement to our implementation would be allowing the parameters of this annealing to be user defined and, thus, less application dependent.

For each of the architectures trained we present the evolution of the validation loss throughout training as well as two metrics for accessing the smoothness of the latent space. The evolution of the loss terms help determine the convergence of the models, the point where additional training yields diminishing returns in performance. The latent space smoothness metrics are indicative of how well that model should perform when employed with a latent space navigation strategy for targeted generation. The two metrics

are similar, both measuring the validity of reconstructed latent points with varying magnitudes of Gaussian noise being added to them, increasing standard deviation. The first metric then evaluates the reconstruction of latent encodings of known molecules, while the second one evaluates the reconstruction of random points in latent space. For VAEs and AAEs with a Gaussian prior, architectures where the latent space is meant to match the standard normal distribution, the two metrics are expected to roughly match. For other architectures, however, while the first metric is still indicative of the resilience of the decoder, the second one is expected to perform significantly worse.

mosesVAE: This VAE was trained for 30 epochs with the reconstruction loss, Figure 43 top, stabilizing after epoch 10 while the latent regularization loss only stabilized after epoch 20. This delay was expected due to the annealing of the second loss term employed during training, allowing the model to first learn to reconstruct its input before regularizing its latent space. From around epoch 5, the total loss was dominated by the latent term, meaning the model had mostly learned how to reconstruct molecules and thereafter attempted to improved the regularization of the latent space. As Figure 43 bottom shows, the model achieved over 90% validity in reconstructing molecules, even when subjected to the largest perturbations. When generating molecules from novel latent points, it attained over 90% validity up to sampling using the standard Gaussian N(0,1), dropping to around 75% for the highest magnitude tested.

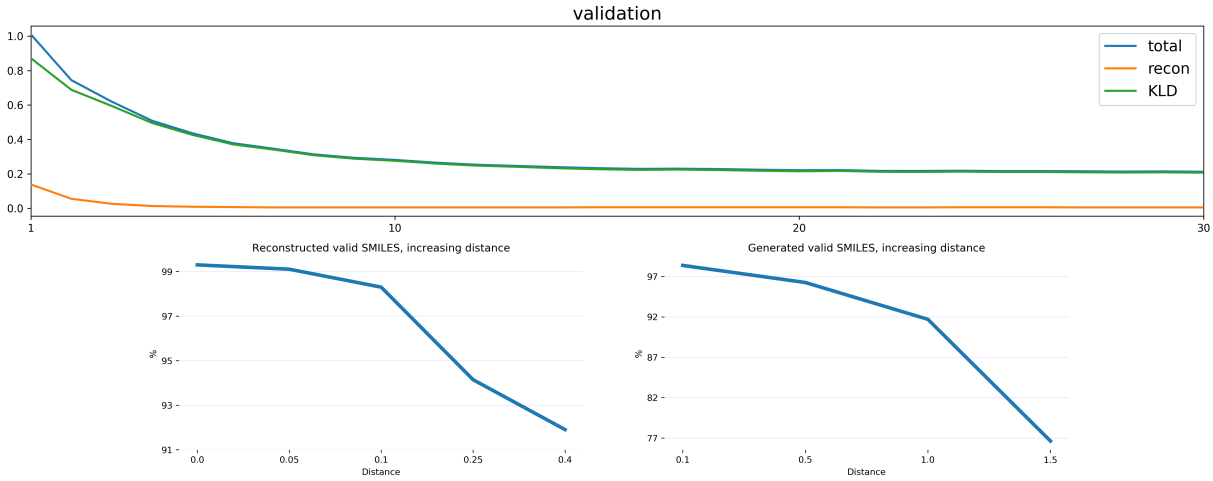


Figure 43: **Top**: Evolution of the validation loss during the 'mosesVAE' model training. The orange line, labeled 'recon', shows the reconstruction loss, the green line, labeled 'KLD', shows the latent regularization loss and the blue line, labeled 'total', shows the sum of the previous two terms. **Bottom**: Evaluation of the latent space smoothness of the trained model. The vertical axis shows the percentage of validly reconstructed molecules while the horizontal axis shows increasing magnitudes of Gaussian noise added. The left graph then shows the reconstruction of points arround known molecules while the right graph shows the reconstruction of random points.

blaschkeVAE: This VAE was also trained for 30 epochs, Figure 44 top, and with annealing of the latent regularization penalty. Its loss terms evolved in a similar way to the 'mosesVAE' model, stabilizing between epoch 10 and 20. However, it presented the worst performance in both the reconstruction and generation tasks, Figure 44 bottom, with the first remaining under 0.5% and the second under 0.35%. These results

seem to indicate that the architecture failed to learn how to generate molecules.

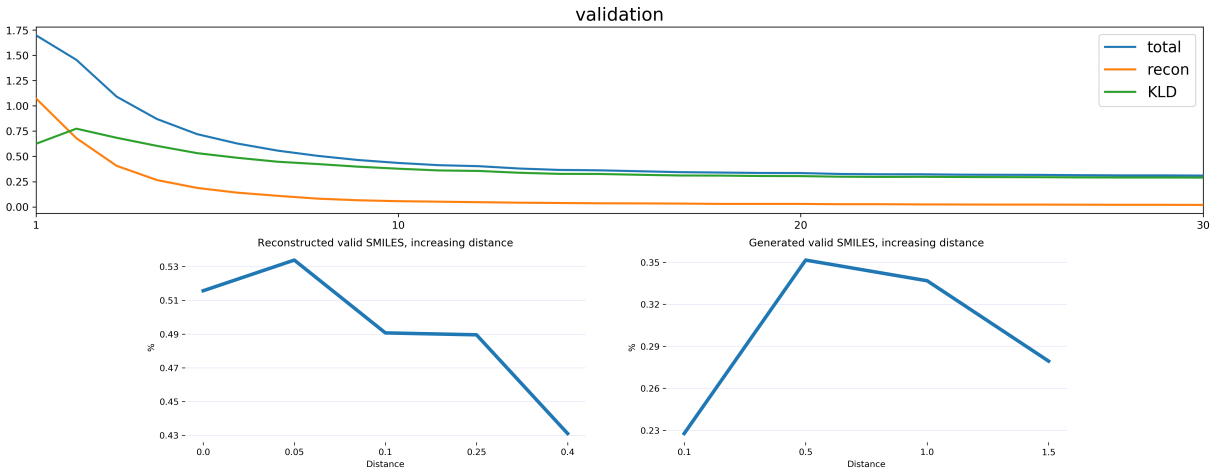


Figure 44: **Top**: Evolution of the validation loss during the training of the 'blaschkeVAE' model. The orange line, labeled 'recon' shows the reconstruction loss, the green line, labeled 'KLD', shows the latent regularization loss and the blue line, labeled 'total', shows the sum of the previous two terms. **Bottom**: Evaluation of the latent space smoothness of the trained model. The vertical axis shows the percentage of validly reconstructed molecules while the horizontal axis shows increasing magnitudes of Gaussian noise added. The left graph then shows the reconstruction of points around known molecules while the right graph shows the reconstruction of random points.

mosesAAE: The AAE was also trained for 30 epochs with its loss terms presenting various spikes and only stabilizing near the end of training, Figure 45 top. This volatility is a common hurdle of adversarial training where, for instance, the discriminators ability to identify synthetic samples may outpace the generator/encoders ability to fool it. Nevertheless, as Figure 45 bottom shows, this model achieved over 99% validity when reconstructing molecules with only a very slight decrease as the distance to the initial point increased. Likewise it attained over 98% validity when generating new compounds, with again only a slight decrease in validity for the higher magnitudes of noise.

seq2seqAE: This last model was allowed to train for 50 epochs with the reconstruction loss, Figure 46 top, stabilizing around epoch 40. As Figure 46 bottom illustrates, the model achieved around 40% valid reconstructions when the latent encoding was subjected to small or no perturbations, with that value falling to under 30% with the increase in magnitude of the Gaussian noise being added. Meanwhile, the sampling of molecules by decoding random latent points varied between 2 and 12%, depending on the standard deviation of the Gaussian noise employed.

6.1.3 *MOSES benchmark*

The Molecular Sets (MOSES) benchmark introduced by Polykovskiy et al. [85] aims to allow the comparison of various molecular generative models. A standard dataset with 1.7M molecules from ZINC[29] is provided

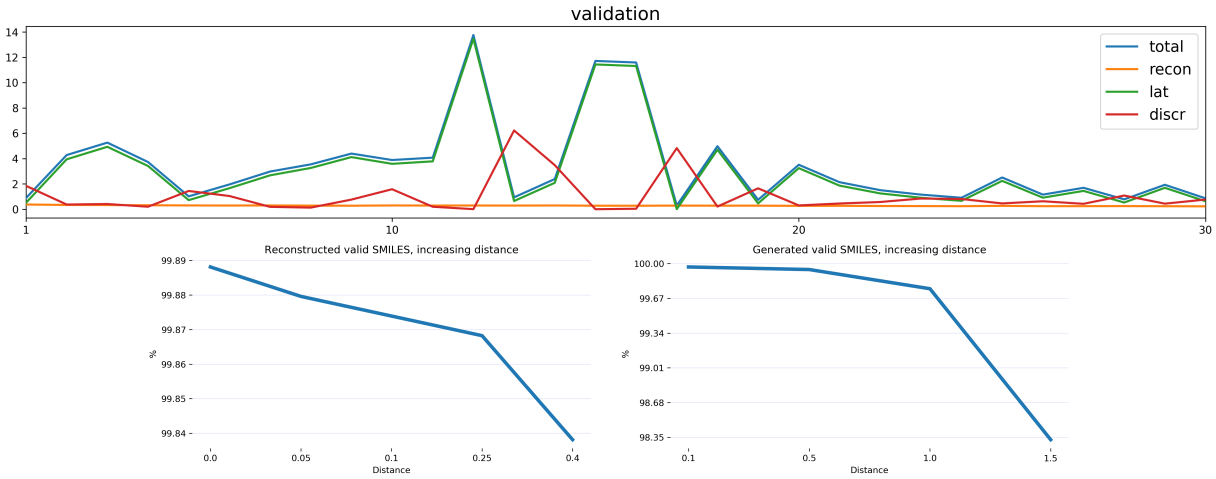


Figure 45: **Top**: Evolution of the validation loss during the 'mosesAAE' model training. The orange line, labeled 'recon' shows the reconstruction loss, the green line, labeled 'lat', shows the latent regularization loss and the blue line, labeled 'total', shows the sum of the previous two terms or rather the loss of the AE part of the model. The red line, labeled 'discr', shows the adversarial training loss of the discriminator and is not included in the total. **Bottom**: Evaluation of the latent space smoothness of the trained model. The vertical axis shows the percentage of validly reconstructed molecules while the horizontal axis shows increasing magnitudes of Gaussian noise added. The left graph then shows the reconstruction of points around known molecules while the right graph shows the reconstruction of random points.

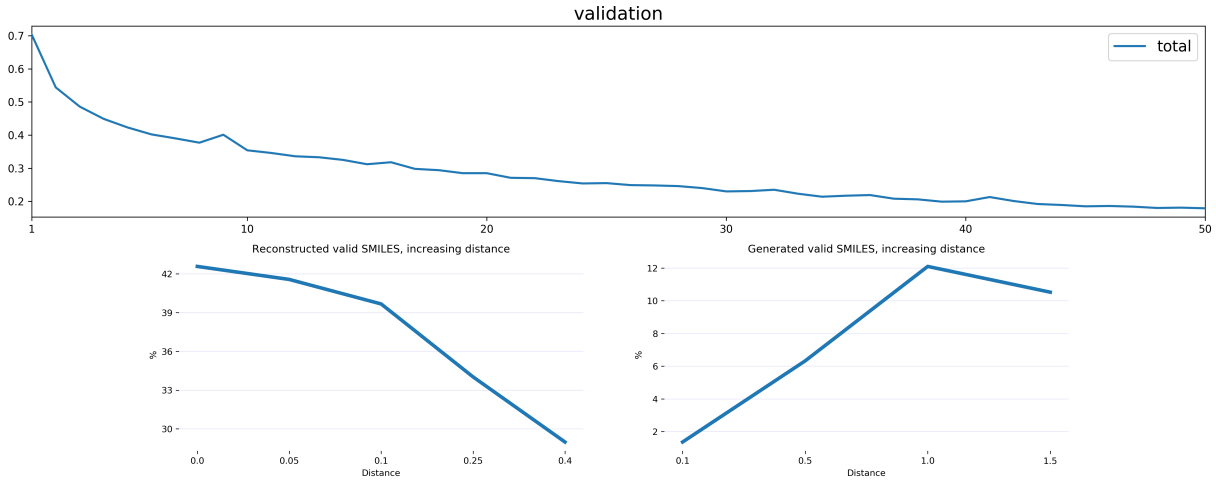


Figure 46: **Top**: Evolution of the validation loss during the 'seq2seqAE' model training. The single blue line shows the reconstruction loss amd is labeled 'total' as this architecture doesn't perform latent space regularization. **Bottom**: Evaluation of the latent space smoothness of the trained model. The vertical axis shows the percentage of validly reconstructed molecules while the horizontal axis shows increasing magnitudes of Gaussian noise added. The left graph then shows the reconstruction of points arround known molecules while the right graph shows the reconstruction of random points.

for performing the training and testing of the models. As described previously, our models were trained on this dataset. This benchmark includes some routine metrics to assess the general quality of the models, such as the fraction of valid and unique generated molecules, their novelty with regards to the training set and their internal diversity. These metrics mainly help to determine the viability of the generative model. It then also provides a set of metrics meant to evaluate how well the model learned features of the training

dataset. For this purpose, it presents the FCD, the Tanimoto Similarity to a nearest neighbor (SNN), the cosine similarity of Bemis–Murcko scaffolds (Scaf) and also the the cosine similarity of BRICS fragments (Frag). The two last metrics help compare molecules at a substructure level, while the first two help capture more abstract chemical and biological similarities.

Table 4 outlines the results of our models, alongside two baselines, in this benchmark. From the general quality metrics, first 6 rows, it can be noted that our implementation of the architectures from Polykovskiy et al.[85] labeled mosesVAE and mosesAAE generally matched the performance of their baselines. Our other two models however, exhibited significantly worse performance in the validity metric. This led to the VAE implemented following Blaschke et al. not being considered for the case study. As the *seq2seq* based AE presented low but manageable validity, while its remaining metrics remained acceptable, it wasn't discarded.

For the metrics relating to matching the training dataset, the mosesVAE and mosesAAE models performed slightly worse than the baselines. Meanwhile, the seq2seqAE and blaschkeVAE models again displayed significantly worse performance compared to the baselines.

These results seem to indicate that further training could result in better scores among the second set of metrics. These improvements however, would also likely be accompanied by a loss of novelty in the generated molecules, as they would more closely follow the training data. As our case study benefited from models with high novelty, these results were deemed acceptable.

Table 4: MOSES benchmark

metric\model	VAE	AAE	mosesVAE	mosesAAE	seq2seqAE	blaschkeVAE
valid	0.9767	0.9368	0.9178	0.9977	0.0893	0.003
unique@1k	1.0	1.0	1.0	0.9953	1.0	1.0
unique@10k	0.9984	0.9973	1.0	0.9669	0.9994	1.0
Novelty	0.6949	0.7931	0.9862	0.5857	0.9996	1.0
IntDiv	0.8558	0.8557	0.859	0.835	0.8925	0.8615
IntDiv2	0.8498	0.8499	0.8537	0.8273	0.8837	0.8207
Filters	0.997	0.996	0.9721	0.9994	0.6372	0.0433
FCD/Test	0.099	0.5555	0.7479	2.2461	17.7413	47.8109
FCD/TestSF	0.567	1.0572	1.1348	2.9069	18.5532	48.6907
SNN/Test	0.6257	0.6081	0.4864	0.6851	0.3252	0.2239
SNN/TestSF	0.5783	0.5677	0.4705	0.6256	0.3191	0.2167
Frag/Test	0.9994	0.991	0.9953	0.9899	0.7877	0.7108
Frag/TestSF	0.9984	0.9905	0.994	0.9864	0.7849	0.6951
Scaf/Test	0.9386	0.9022	0.8069	0.8433	0.0091	0.0
Scaf/TestSF	0.0588	0.0789	0.1788	0.0303	0.0015	0.0

6.2 Validation of the EAMO Framework

Next, seeking to validate the EAMO framework we employed it in a set of case studies used in recent literature [66, 99, 103, 110, 116]. To keep the process manageable, these case studies were executed with a single architecture. As such, we employed the mosesVAE model and, in accordance to the discussion above, elected to use the checkpoint from epoch 20 as the model seemed to stabilize thereafter. Except for the similarity constrained case study, the initial population was randomly selected from the test set of the dataset used for training the model. We now define each of these case studies, followed by presenting and discussing the results obtained. We focus on comparing our framework with other state-of-the-art methods while also contrasting the SO and MO approaches possible under EAMO.

6.2.1 *Case Studies*

As a first step, we started by optimizing for a single molecular property, the QED[146]. Then, we targeted the "penalized solubility" or PlogP. Next, this objective was also optimized, while constrained to be similar to an initial set of molecules, and finally, we targeted a more pragmatic objective by jointly optimizing the predicted activity towards the DRD2, the QED, and the predicted non-toxicity. Next, we further detail each of the case studies.

Diversity

Throughout some of the case studies we also discuss the diversity of the molecules being generated. As defined by [103], we considered molecular diversity as the average pairwise Tanimoto distance between each molecule in given set of compounds, as follows:

$$diversity_A = \frac{\sum_{x,y \in A} 1 - sim(x,y)}{|A|} \tag{2}$$

where $sim(x,y)$ is the Tanimoto similarity between the Morgan Fingerprints [151] with radius 2 of two molecules.

QED

One of the most valuable metrics considered during the early stages of drug discovery, the QED[146] combines various molecular properties into a single score. Specifically, it takes into account various of molecular properties, including molecular weight, logP, TPSA, number of hydrogen bond donors and acceptors, the number of aromatic rings and rotatable bonds, and the presence of unwanted chemical functionalities. This score is often indicative of whether a new compound will pass to later stages of drug discovery [146]. As illustrated in Figure 33, it is computed from the molecular structure using the open-source cheminformatics software RDKit[37].

Solubility penalized by synthetic accessibility and large rings (PlogP)

The octanol-water partition coefficient serves as a measure between lipophilicity (fat solubility) and hydrophilicity (water solubility) of a molecule. Molecules with a positive logP are lipophilic, while molecules with a negative logP are hydrophilic. The PlogP is defined as:

$$PlogP(m) = loqP(m) - SA(m) - cycle(m) \tag{3}$$

where $SA(x)$ is the molecule synthetic accessibility score and $cycle(m)$ counts the number of atoms over six in the largest ring. In the MO optimizations, each term in Eq. 3 is taken as a separate objective.

Penalized solubility constrained by similarity to set of molecules

The constrained PlogP extends the previous by enforcing a similarity constraint to a given starting molecule, that is, $sim(x,y) \geq \delta$. The experiments consider a set of 800 molecules obtained from [103] (selected for having the worst score in their dataset) and constrains $\delta \in \{0, 0.2, 0.4, 0.6\}$. Similarly to the computation of diversity in equation 2, $sim(x,y)$ is taken as the Tanimoto similarity between the Morgan Fingerprints with radius 2 of molecules x and y. The constraint, formulated as a new optimization objective, is defined as:

$$constraint = \begin{cases} sim(x,y) + 200, & \text{if } sim(x,y) \geq \delta \\ sim(x,y) * 100, & \text{otherwise} \end{cases} \tag{4}$$

where δ is the defined threshold and $sim(x,y)$ is the aforementioned similarity between the solution being considered and a given starting molecule. This was designed to provide a clear signal, both guiding the algorithm towards meeting the constraint when it is not yet met and distinctively marking when the constraint has been satisfied.

DRD2 activity and drug-likeness penalized by toxicity

This case study's main goal is to find molecules with increased activity against a biological target, the DRD2. To assess molecules' activity probability, we use the SVM classifier with a Gaussian kernel proposed in [63].

Two additional objectives are considered, the QED and the non-toxicity of the compound. The last one is predicted using a Multi-Target DNN (MT-DNN) trained in house on the Tox21 dataset [33]. With this model, a molecule is considered toxic, and given an evaluation of 0.0, if it scores over 0.5 in two or more endpoints. However, if it does not score over 0.5 for any endpoint, it is considered non toxic and evaluated as 1.0. When a molecule scores over 0.5 in one or two endpoints, it is evaluated with the average of the scores.

6.2.2 *Results and Discussion*

In this next section, we report and discuss the results for each of the case studies. Each case study took on average 4 hours to fully execute, with the exception of the constrained PlogP that required close to 100 hours for the higher thresholds. All experiments were run without parallelization as only one GPU was available.

QED

Optimizing QED serves as a first test of whether our method can effectively optimize molecules in the latent space of a chemical AE. For this task, we performed SO optimization with a GA, stopping after 500 generations and executing 30 runs. As with previous methods, we report the 3 best molecules found and the 95% confidence interval for the mean of the best score in each of the 30 runs. Our results alongside the baselines are outlined in Table 5.

Most previously reported methods already saturated this score, plateauing at 0.948. Likewise, our framework achieved a very similar performance, with the worst of all runs scoring 0.9475. Furthermore, the best molecules found between all runs were also novel and diverse, featuring 0.812 calculated diversity. These results, therefore, indicate that our framework can indeed optimize abstract molecular properties while also generating several distinct solutions.

Table 5: Baselines and our results for the optimization of QED reporting the scores for the three best molecules. (* is reported by You et al [99])

Method	**QED**			
	1st	**2nd**	**3rd**	**Mean**
Guimaraes et al. [10]*	0.896	0.824	0.820	-
Popova et al. [95]	0.948	0.948	0.947	-
Jin et al. [103]*	0.925	0.911	0.910	-
You et al. [99]	0.948	0.947	0.946	-
Winter et al. [110]	0.948	-	-	-
Ours (SO)	0.948	0.948	0.948	0.9481 ± 0.0001

PlogP

The PlogP aimed to be a more realistic and demanding objective, seeking to produce molecules that are not only highly hydrophobic, but also easy to synthesise and lack unusual ring structures. However, as reported previously [99, 110], this objective can be trivially solved with long aliphatic chains that lack rings and are hydrophobic and easily synthesised.

For this task, we performed both SO and MO optimization with a GA and NSGA-III, respectively. The algorithms were allowed to run for 500 generations and were executed 30 times. Like previous methods, we report the 3 best molecules found and the 95% confidence interval for the mean of the best score in

each of our 30 runs. Our results, presented in Table 6, were similar to [110], with optimization yielding long aliphatic chains.

Besides the jarringly high final scores, a brief comparison can be established between the SO and the MO case. In the first, out of the 30 runs, the algorithm converged to the same exact molecule in all but 2, resulting in a mean diversity among runs of 0.007. In contrast, the MO case found several different high scoring compounds, and each run resulted in large sets of unique molecules, with a mean diversity among runs of 0.44.

As our scores were consistent with previously reported work, we determine that our framework could successfully complete this case study and even achieving higher absolute values. Furthermore, this experiment also highlighted distinct performances between SO and MO, with the later achieving slightly superior best scores while maintaining a higher diversity of solutions.

Table 6: Baselines and our results for the optimization of PlogP. ([a]reported in an earlier version of their work; *reported by You et al. [99])

Method	PlogP			
	1st	2nd	3rd	Mean
Gómez-Bombarelli et al. [66][a]	5.02	4.68	-	-
Kusner et al. [72]	2.94	2.89	2.80	-
Dai et al. [102]	4.04	3.50	2.96	-
Jin et al. [103]	5.30	4.93	4.49	-
Samanta et al. [104]	2.83	2.48	2.30	-
Winter et al. [110]	26.1	-	-	-
Popova et al. [95]	10.34	10.19	10.14	-
You et al. [99]	7.98	7.85	7.8	-
Guimaraes et al. [10]*	3.63	3.49	3.44	-
Bresson and Laurent [105]	5.24	5.10	5.06	-
Griffiths et al. [101]	4.01	-	-	-
Ours (SO)	32.21	22.01	-	22.35 ± 0.67
Ours (MO)	32.57	25.74	24.30	22.64 ± 0.73

Constrained PlogP

By constraining solutions to be similar to a specific starting molecule, the trivial result discussed previously is no longer feasible. Therefore, this case study is inherently more challenging as it includes an additional objective (Eq. 4), while remaining a relevant problem. The added difficulty results from drastically reducing the number of viable solutions, thus hindering the optimization process. Still, it is of great importance as optimizing a molecule to be similar to a given one is a common problem faced in drug design.

For this task, each of the 800 compounds was independently optimized for each of the thresholds. Furthermore, an "early" stopping criterion was also employed where, after a minimum of 250 generations, the optimization was allowed to end once a valid solution was found or reach 2500 generations, otherwise.

Following previous work, we report the success rate, the percentage of molecules for which a valid improvement was found, and within those, the mean and standard deviation of the best improvement and

corresponding similarity to the initial molecule. As such, only methods with similar success rates should be directly compared. Table 7 outlines our results and Table 8 the baselines. For both the first and second threshold, 0.0 and 0.2, we outperformed all baselines while maintaining equivalent success rates. Of note, the threshold of 0.0 is identical to the unconstrained case, and, as such, the trivial solution emerges again. For the threshold of 0.4, our success rate is placed between the methods of Bresson and Laurent [105] and Maziarka et al. [108], surpassing both in terms of average improvement. The last threshold of 0.6 has a success rate just slightly below that of the approach of Maziarka et al. [108], but with a bigger average improvement.

Table 7: Our results for the constrained PlogP case study, using either single-objective or multi-objective algorithms, with 2500 maximum generations.

	Ours (SO)			**Ours (MO)**		
δ	**Improvement**	**Similarity**	**Success**	**Improvement**	**Similarity**	**Success**
0.0	25.73±2.43	0.02±0.02	100.0%	25.91±1.99	0.02±0.01	100.0%
0.2	7.50±1.85	0.22±0.03	98.37%	6.96±2.18	0.23±0.03	99.75%
0.4	4.04±1.53	0.42±0.02	72.12%	3.81±1.63	0.43±0.03	67.12%
0.6	2.42±1.41	0.62±0.02	17.25%	2.39±1.34	0.63±0.03	17.87%

Table 8: Baselines for success rates and improvements in the constrained PlogP case study.

	Bresson and Laurent [105]			**Maziarka et al.** [108]			**You et al.** [99]		
δ	**Improv.**	**Similarity**	**Success**	**Improv.**	**Similarity**	**Success**	**Improv.**	**Similarity**	**Success**
0.0	5.24±1.55	0.18±0.12	100.0%	8.30±1.98	0.16±0.09	99.75%	4.20±1.28	0.32±0.12	100.0%
0.2	4.29±1.57	0.31±0.12	98.60%	5.79±2.35	0.30±0.11	93.75%	4.12±1.19	0.34±0.11	100.0%
0.4	3.05±1.46	0.51±0.10	84.0%	2.89±2.08	0.52±0.10	58.75%	2.49±1.30	0.47±0.08	100.0%
0.6	2.46±1.27	0.67±0.05	40.1%	1.22±1.48	0.69±0.07	19.25%	0.79±0.63	0.68±0.08	100.0%

A separate metric to consider in this case study, as introduced by [116], is the diversity of the valid solutions found for each molecule. Table 9 outlines the results and baselines used in [116] alongside our results. The diversity attained by our methods is slightly worse but still in line with the baseline. Furthermore, the MO case, again, mostly outperformed the SO case in this regard, except for the 0.6 threshold.

Table 9: Results and baselines used in [116] and our results regarding the diversity of solutions for each molecule in the target set.

δ	**VJTNN Jin et al.** [116]	**MMPA baseline**	**VSeq2Seq baseline**	**Ours (SO)**	**Ours (MO)**
0.0	-	-	-	0.334	0.722
0.2	-	-	-	0.553	0.655
0.4	0.480	0.496	0.471	0.416	0.451
0.6	0.333	0.329	0.331	0.298	0.256

The last consideration is the stopping criterion. Figure 47 illustrates the mean number of generations each algorithm took for each threshold. A positive correlation trend is apparent, with higher thresholds

requiring, on average, more generations to find viable solutions. This result is expected as the higher the threshold, the more it reduces the viable solution space, and therefore optimizing gets more challenging. Although no difference is evident between SO and MO in terms of generations used, the first achieved slightly better mean improvement with a higher or comparable success rate. We attribute this to our implementation of the constraint which, in the MO case, allowed the algorithm to focus on invalid solutions as it operated as an independent objective. Meanwhile, in the SO case, solutions not satisfying the constraint were heavily penalized, helping the algorithm to focus on viable solutions.

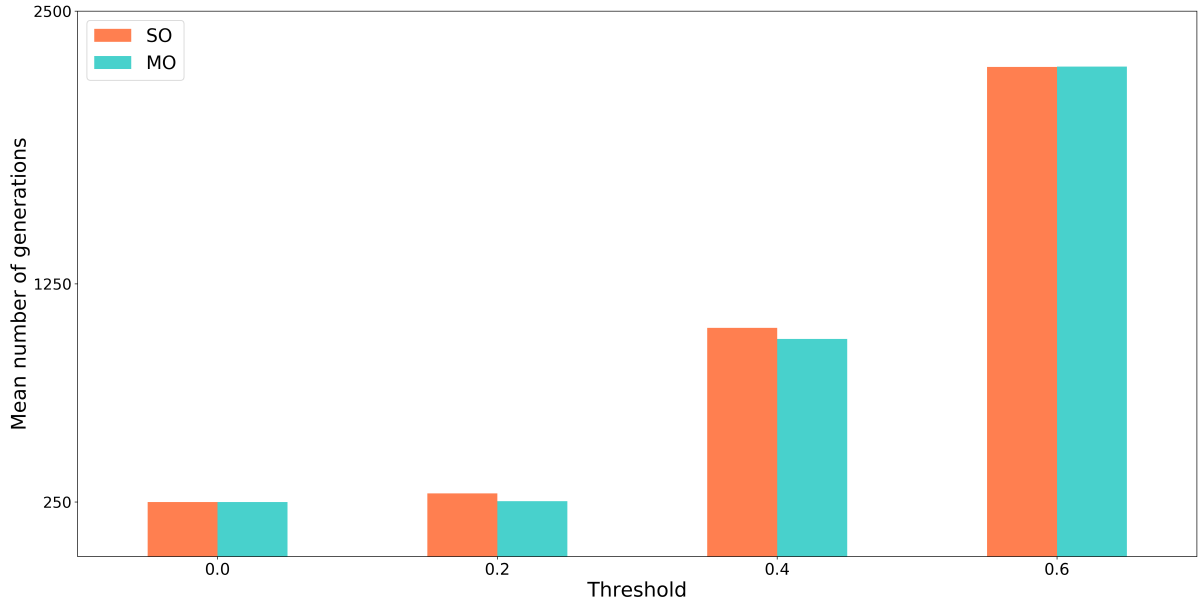


Figure 47: Mean generations taken by the SO and MO for each of the thresholds. The minimum was set as 250 and the maximum 2500. Should the algorithm find a viable improvement during this interval it was allowed to terminate.

DRD2 activity and Drug-likeness penalized by toxicity

The DRD2 activity problem encompasses the simultaneous maximization of three objectives: DRD2, QED, and non-toxicity (nonTox). To address this problem, we considered a GA and three commonly used MOEAs: NSGA-III, SPEA2, and GDE3. Besides allowing the evaluation of the proposed approach in a more pragmatic problem, we also aim to compare the performance of the EAs. For the GA, the objective function is the aggregated sum of all three objectives, which is also used as a comparison score in Table 10. The values for DRD2, QED and nonTox vary in the range $[0, 1]$. We show the 95% confidence interval for the mean of the best aggregated score along with the value of its respective individual objectives for the 30 repetitions.

The results indicate that GDE3 performs best with the highest aggregated score. However, this performance does not extend to the diversity metric, where GDE3 delivered molecules with the lowest diversity. On the other hand, the GA found molecules with the overall worst score but with the highest diversity. To further assess the MOEAs performance in this optimization task, we include in Table 11 three of the most commonly used Quality Indicators [152], notably, the *Hyper Volume* which measures the size of the

Table 10: Optimization results for DRD2 activity and QED penalized by toxicity.

Algorithm	Score	DRD2	QED	nonTox	Diversity
GA	2.877 ± 0.026	0.969 ± 0.020	0.908 ± 0.014	1.0 ± 0.0	0.775
NSGAIII	2.882 ± 0.031	0.960 ± 0.020	0.922 ± 0.014	1.0 ± 0.0	0.754
SPEA2	2.902 ± 0.014	0.976 ± 0.012	0.926 ± 0.009	1.0 ± 0.0	0.748
GDE3	2.935 ± 0.002	0.995 ± 0.002	0.940 ± 0.002	1.0 ± 0.0	0.519

portion of objective space that is dominated by a set of solutions, a Pareto Front (PF); the *Generational Distance* which is used to access the convergence of the solution by measuring the distance from the solution PF to the best PF approximation determined by combining non-dominated points from all MOEA solutions; and the *Epsilon (ϵ)* indicator which measures the smallest value to be added to any point in the solution PF to make it non-dominated by some point in best PF approximation. All three indicators point to a better performance of GDE3.

Table 11: Quality indicators for the MO-EAs

Algorithm	ϵ-indicator	Generational Distance	Hypervolume
NSGAIII	0.0446	0.1127	0.9111
SPEA2	0.0359	0.1188	0.9356
GDE3	0.0082	0.0927	0.9451

6.2.3 *Conclusions*

With the results from the previous experiments, it is apparent that the proposed framework is capable of controlling abstract chemical properties of generated molecules, as well as to perform tasks relevant to molecular design, such as optimizing given molecules and generating possible leads for biological receptors. Regarding the scoring on the several benchmarks from the state-of-the-art, the framework mostly outperformed, or at least matched, results from previous work, presenting itself as a viable alternative. Throughout the various experiments, the use of MO algorithms often led to more diverse solution sets. This is an important advantage over the SO case as having a diverse set of leads is paramount for early *de novo* chemical design. Lastly, a better implementation of the similarity constraint in the constrained PlogP case study may enable the MO algorithm to match, and even surpass, the performance of SO.

6.3 Generating Novel Sweeteners

Having validated both our frameworks and methods for the targeted generation of new compounds, we sought to tackle the interesting application of generating new sweeteners. For this, we employed the models discussed above with the exception of 'blaschkeVAE' due to its poor performance. To guide the generative process, the models were fine-tuned on a set of known sweeteners and also combined with EAs through EAMO.

For this process, we also employed ML classifiers, specifically an ensemble combining a SVM, a RF and a DNN, to predict the sweetness of the generated compounds. Furthermore, the MT-DNN discussed in Chapter 6.2.2 was also employed to help ensure the new molecules were non-toxic. All of these models had already been trained and validated in previous work done by the research group.

This process resulted in a first set of novel predicted sweeteners which were reviewed by expert chemists collaborating under a research project. With the feedback from this set, we then improved on our methods and produced a second set of new molecules. The specifics of this case study are further discussed below.

6.3.1 *Transfer Learning*

Dataset

As discussed during Chapter 4.4.2, transfer learning requires data with the desired properties with which to bias the model. In this instance, the target characteristic is sweetness and thus a dataset of known sweeteners was employed. This dataset was already available within the research group and, as Table 12 shows, was compiled from 9 different sources. Compounds for these sources were standardized and combined, resulting in 2529 unique molecules of which 1315 were labeled as sweet and 1214 as non-sweet. The discrepancy between the total number of molecules gathered and the final set was due to the removal of duplicates.

Table 12: Compilation of the known sweeteners dataset

Source	**Sweet**	**Non-sweet**
Tuwani et al. [153]	1205	1171
Dagan-Wiener et al. [154]	-	107
Banerjee et al. [155]	517	685
Rojas et al. [156]	435	214
Zhong et al. [157]	435	214
Ojha et al. [158]	239	-
Chéron et al. [159]	316	-
Garg et al. [160]	291	-
Belitz et al. [161]	210	-
Total unique	1315	1214

Training and Evaluation

With an adequate set of molecules at hand, the models previously discussed were fine-tuned towards generating molecules similar to known sweeteners. For this, we leveraged the 'fine-tuning' task implemented in the DeepMolGen framework. The training parameters were adjusted to better suit the small dataset with the batch size being reduced to 32 and the learning rate to 0.0005. Furthermore, to allow more granularity when selecting the final model for use, checkpoints were created after each epoch. Lastly, each model was allowed to fine-tune for a total of 20 epochs.

For each of the architectures fine-tuned we present the evolution of the loss throughout training, as well as four metrics for assessing the progress of transfer learning. Due to the already small size of the dataset, a validation set was not created. Instead, the presented 'validation' loss was obtained from the known full data at the end of each epoch. For the metrics, the models were sampled after each epoch, and the evolution of the validity, predicted sweetness, novelty and uniqueness plotted.

mosesVAE: As with Chapter 6.2, we elected to employ the checkpoint of epoch 20 as the starting point to fine-tuning. As Figure 48 top shows, during the transfer learning process, there was mainly a gradual reduction of the latent loss term, with little change to the reconstruction loss. This was the expected outcome as the model had already learned to generate valid molecules and this process should mainly bias it towards generating more sweeteners. Indeed already after the first epoch, around 20% of the generated molecules were predicted as sweet. This then continued to improve, with the later epochs yielding close to 60% predicted sweeteners with a negligible impact on the validity of molecules being generated. However, a small increase in the generation of duplicates was observed starting around epoch 10. This remained manageable and didn't affect the usage of the model as it had only reached 3% by epoch 20. The last checkpoint, epoch 20, was chosen for usage as it presented the highest percentage of predicted sweeteners and still presented a minimal impact on the performance of the model.

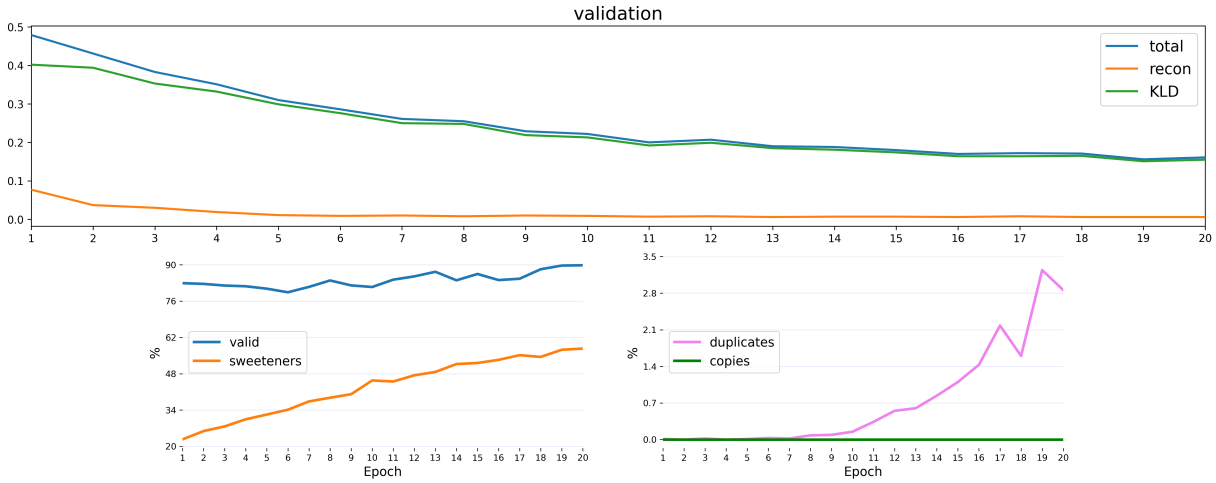


Figure 48: **Top**: Evolution of the validation loss during the 'mosesVAE' transfer learning. The green line, labeled 'KLD', shows the latent regularization loss, the orange line, labeled 'recon', shows the reconstruction loss and the blue line, labeled 'total', shows the sum of these two terms. **Bottom**: Metrics tracking the quality of the model throughout the fine-tuning process. The left graph shows the validity and predicted sweetness of generated molecules. Meanwhile the right graph shows the uniqueness measured as the percentage of 'duplicates' and the novelty measured as direct 'copies' from the dataset.

mosesAAE: For the starting point of the AAE, the last checkpoint was chosen as the validation loss only seemed to stabilize near the end of the allowed 30 epochs. Similar to its initial training run, Figure 49 top shows that the transfer learning process was also marked by some instability. Nevertheless, it led to a very rapid increase in predicted sweetness, reaching 70% within just 2 epochs. This was accompanied by a negligible impact on validity but, crucially, a very significant increase in duplicates being generated. This seemed to indicate the occurrence of mode collapse, a common problem in adversarial training where

the model devolves to generating a single, or very reduced set, of outputs. In an attempt to sidestep this issue, the second checkpoint, epoch 2, was chosen for usage as it already presented a large percentage of predicted sweetness.

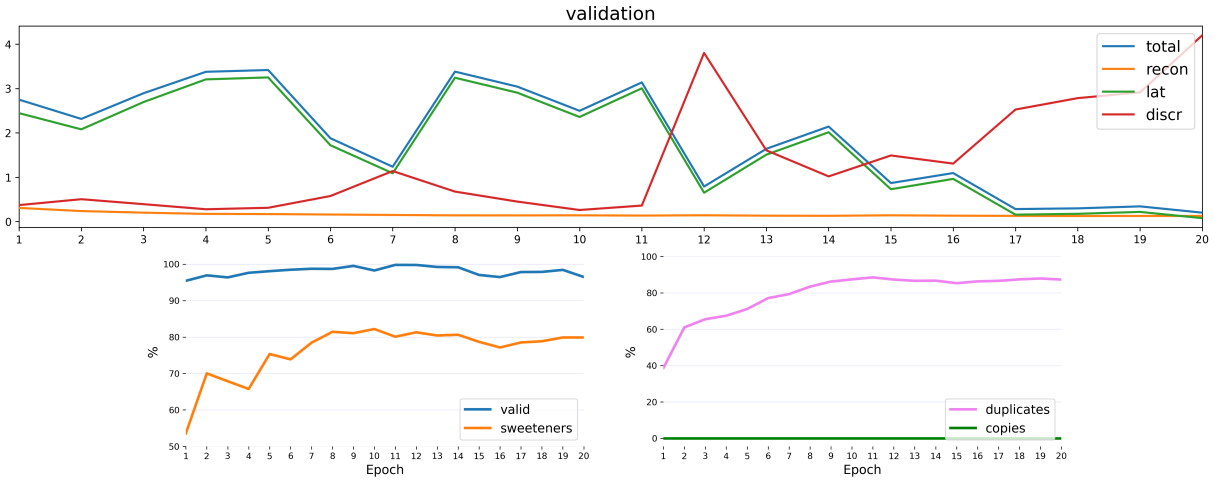


Figure 49: **Top**: Evolution of the validation loss during the 'mosesAAE' transfer learning. The orange line, labeled 'recon' shows the reconstruction loss, the green line, labeled 'lat', shows the latent regularization loss and the blue line, labeled 'total', shows the sum of the previous two terms or rather the loss of the AE part of the model. The red line, labeled 'discr', shows the adversarial training loss of the discriminator. **Bottom**: Metrics tracking the quality of the model throughout the fine-tuning process. The left graph shows the validity and predicted sweetness of generated molecules. Meanwhile the right graph shows the uniqueness measured as the percentage of 'duplicates' and the novelty measured as direct 'copies' from the dataset.

seq2seqAE: The prior seq2seqAE chosen was the second to last checkpoint, epoch 40, as the model seemed to have stabilized at that point. The transfer learning process, Figure 50 top, proceeded as expected, with the majority of the improvements happening before epoch 10. This is also apparent in the validity metric, where a slow increase, culminating in epoch 8 slightly over 11%, was observed after the initial decrease to around 6% from the priors 9%. A large initial increase to 65% of the predicted sweetness was also observed, with the value stabilizing thereafter. Lastly, despite the first impression of the plotted duplicates, these never exceeded 0.15% of the generated compounds. These results then led to epoch 9 being chosen for use mainly due to its validity, the second highest at over 11%.

6.3.2 *Generation of the first set of putative sweeteners*

For this task we employ the 'mosesVAE', 'mosesAAE' and 'seq2seqAE' models. These were respectively trained for 20, 30, and 40 epochs and then also fine-tuned an additional 20, 2, and 9 epochs. Each model was employed within EAMO, both before and after being fine-tuned. Furthermore, the fine-tuned models were also directly sampled. Figure 51 outlines the approach.

EAMO operated in its MO mode leveraging the NSGA-III algorithm. Four distinct objectives were independently optimized namely, the predicted non-toxicity, the predicted natural product likeness, the predicted sweetness and a score for fitting the molecular weight range between 300 and 900 Da. The initial popula-

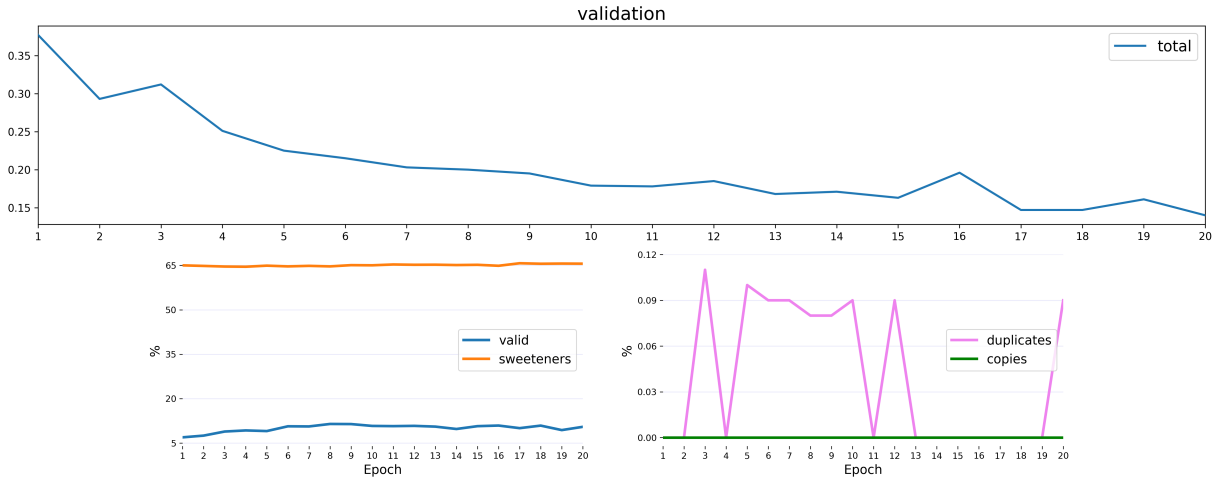


Figure 50: **Top**: Evolution of the validation loss during the 'seq2seqAE' transfer learning. The single blue line labeled 'total' shows the reconstruction loss. **Bottom**: Metrics tracking the quality of the model throughout the fine-tuning process. The left graph shows the validity and predicted sweetness of generated molecules. Meanwhile the right graph shows the uniqueness measured as the percentage of 'duplicates' and the novelty measured as direct 'copies' from the dataset.

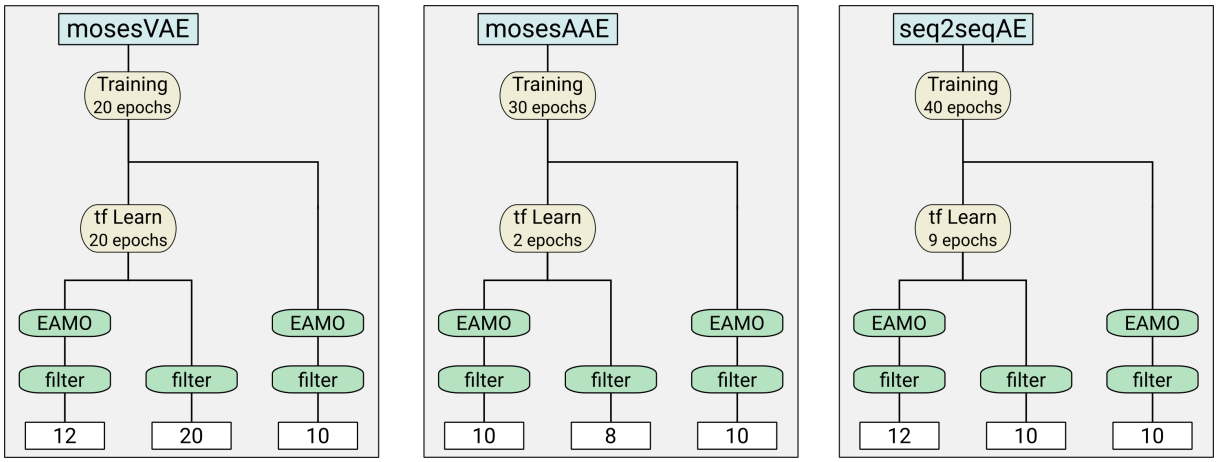


Figure 51: Pipeline used for first sweetener set. The three architectures were trained on a 1.7M subset of the ZINC[29] database and applied within EAMO. The models were then fine-tuned on the known sweeteners and both directly sampled and used within EAMO. The generated molecules then passed a series of filtering steps resulting in a final set of 102 new compounds.

tion was seeded from the known sweeteners dataset, with half the individuals having their genome shuffled to promote novel solutions.

A total of around 100 molecules was expected by our expert chemist partners. As such, approximate quotas for the different combinations of models and methods were defined. Particularly, we aimed at 10 molecules for each of the prior models combined with EAMO, 12 for the fine-tuned models combined with EAMO, and 10 for the direct sampling of the fine-tuned models. To ensure we reached the target of 100 compounds, while allowing for some tolerance in the defined quantities, sampling of the transfer learned 'mosesVAE' model was elected to contribute with a total of 20 total molecules. The model was chosen due to it outperforming the 'mosesAAE' when using transfer learning, producing less duplicates, and the 'seq2seq2AE' in terms of validity and prior model performance.

With this in mind, the 'mosesAAE' and 'seq2seq2AE' fine-tuned models were sampled for 10240 molecules, while the 'mosesVAE' generated an extra 5120. For the EAMO method, the EA was run for 50 generations with a population of 100 individuals. This population was saved at the end of each generation, yielding 5000 molecules for each of the 6 models.

All of these sets of compounds were checked for invalid or duplicated molecules and then filtered for molecular weight, predicted non-toxicity, natural product likeness and sweetness. The sets were then reduced to have a maximum pairwise Tanimoto similarity of 0.2 through a randomized selection process. A visual inspection then followed removing clearly unsuitable compounds, mainly due to unusual ring structures. Direct copies from known sweeteners were only removed last due to oversight and from the resulting sets, the quotas were filled with the molecules with the highest score for predicted sweetness.

Table 13 shows the number of molecules after each of the steps. Of particular interest, the fine-tuned 'mosesAAE' failed to meet its quota due to containing copies of known sweeteners. However, due to the stochastic nature of the similarity selection process and had the removal of direct copies been performed prior to it, this model might have met its quota.

Table 13: Molecules remaining after each step, for each model/method combination, when generating the first set of sweeteners

Method	Model	Generated	Unique	Valid	Filter	Sim.	Visual	Copies	Final
EA	mosesVAE	5000	881	770	17	10	10	10	10
	mosesAAE	5000	4678	4676	40	19	19	19	10
	seq2seqAE	5000	2447	1116	58	10	10	10	10
TfL+EA	mosesVAE	5000	577	529	84	12	12	12	12
	mosesAAE	5000	1367	1319	95	16	13	11	10
	seq2seqAE	5000	3801	2305	291	20	20	20	12
TfL	mosesVAE	15360	14799	13313	50	20	20	20	20
	mosesAAE	10240	4007	3794	111	19	12	8	8
	seq2seqAE	10240	10238	1154	11	11	10	10	10

Interestingly, while the observed values of uniqueness and validity when directly sampling from the models neatly reflected the performance detailed previously, these metrics behaved differently under the EA. Specifically, when combined with the EA the uniqueness dropped in all the models. Meanwhile, the validity either remained high or, in the case of the 'seq2seqAE' model, improved. Going from 11% to 60% for the fine-tuned model and from 9% to 45% for the prior model. This first change, in uniqueness, can be attributed to the fact that the final set combined multiple generations of a single population and as such close similarity, or even replicas, was expected. The improved validity can be attributed to the EA focusing in well behaved regions of latent space, as individuals encoding invalid molecules had their scores heavily penalized.

6.3.3 *Feedback from experts*

The final set of molecules were then carefully considered by expert chemists, handpicking a select few for synthesis and providing short justifications in case of rejection. Figure 52 shows the feedback received. Of the 102 molecules, 73 were rejected and 16 were accepted for further consideration. Furthermore, a third set, labeled as 'maybe', included molecules that could be considered pending some small modifications.

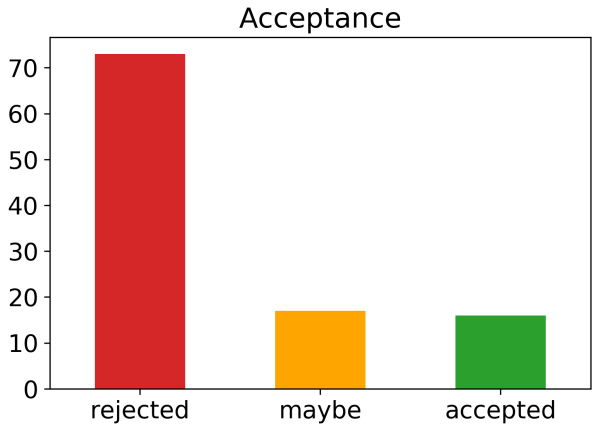


Figure 52: Classifications given by expert chemists to the molecules generated in the first set of sweeteners. Of the 102 molecules, 73 were rejected, 16 were accepted for further consideration and 17 required modifications.

Along with the mentioned feedback, the partners also provided some general improvement guidelines detailing the main concerns they faced when reviewing the set of molecules. Specifically:

- Improve water solubility;
- Reduce number of possible stereoisomers;
- Keep ring sizes between 4 and 7;
- Favor heterocycles with less substitutions;
- Avoid Poliphenols and Phenylethilamines.

To help analyze the reasons that lead to rejections, the short descriptions were binned into 7 groups and further segmented by generative model. As Figure 53 shows, SA was the main cause for rejection, followed by undefined chirality in molecules with many chiral centers. It is worth noting, however, rejections due to SA included various root causes such as poor water solubility or unusual sub-structures. Interestingly, when looking at the rejection causes segmented by model, we can observe that not all models commit all types of errors. For instance, only the 'mosesAAE' produced the possibly toxic phenylethilamine sub-structure. Meanwhile, it led to no rejections due to unusual rings and heterocycles. Likewise, the 'seq2seqAE' didn't produce polyphenols or molecules with undefined chirality. This is in accordance with reports from the state-of-the-art pointing out that different architectures often cover distinct regions of chemical space [107].

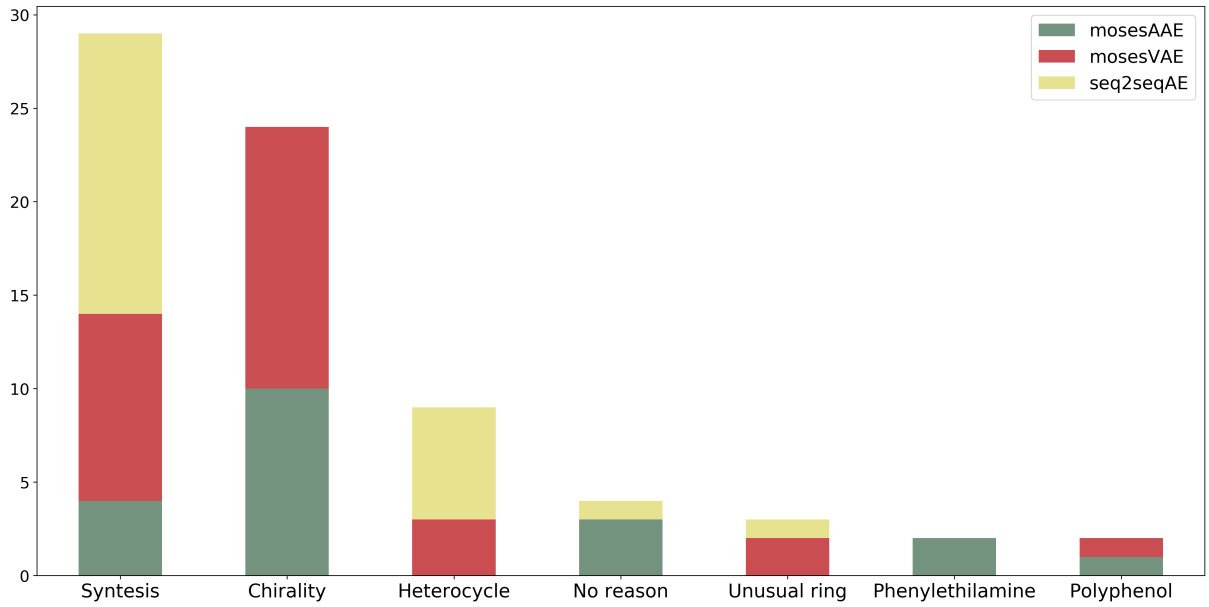


Figure 53: Reasons for rejection in the first sweeteners set, binned into 7 groups and further segmented by generative model. The 'Synthesis' label pertains to molecules which would be difficult to synthesize, the 'Chirality' label to ones with too many possible conformations, the 'Heterocycle' label to ones with cycle with too many substitutions, the 'no reason' label pertains to rejected molecules for which no justification was provided, the 'Unusual ring' label to rings that were either too large or to small and the 'phenylethilamine' and 'polyphenol' labels correspond to molecules containing that respective substructure.

Next, the outcome from each molecule was traced back to the respective generative model and method used to create it. Doing so allows the assessment of the models performance from an alternative, well informed, perspective on its practical usefulness. Figure 54 shows these results. By looking at the acceptance rate of each model, we can conclude that the 'mosesAAE' was the model with the highest positive outcome rate. Furthermore, transfer learning lead to a decrease in acceptance when compared to the prior models. In the case of the 'seq2seqAE' model, no molecules generated after fine-tuning were accepted. This seems to indicate that the increase in predicted sweetness observed when fine-tuning the models, was also accompanied by diminishing molecular quality.

These results thus seem to support the application of the prior models under EAMO and disfavor the combination of both transfer learning and EAs. Furthermore, the 'mosesAAE' model can be seen to outperform the others, where even after fine-tuning, and with limited uniqueness, it came close to the acceptance rate of the 'seq2seqAE' prior to transfer learning. Nonetheless, as discussed previously, some evidence exists for different architectures producing different types of chemistry with our results, Figure 53, pointing in the same direction. As such, despite transfer learning hindering the acceptance rate, and the 'mosesAAE' model outperforming the other models, all these should still be included to improve chemical diversity. A more reasonable approach thus could be to adjust the quotas for each method and model in accordance to these results.

To help assure the quality of future generated molecules, new filtering steps were implemented, specifically, selecting for:

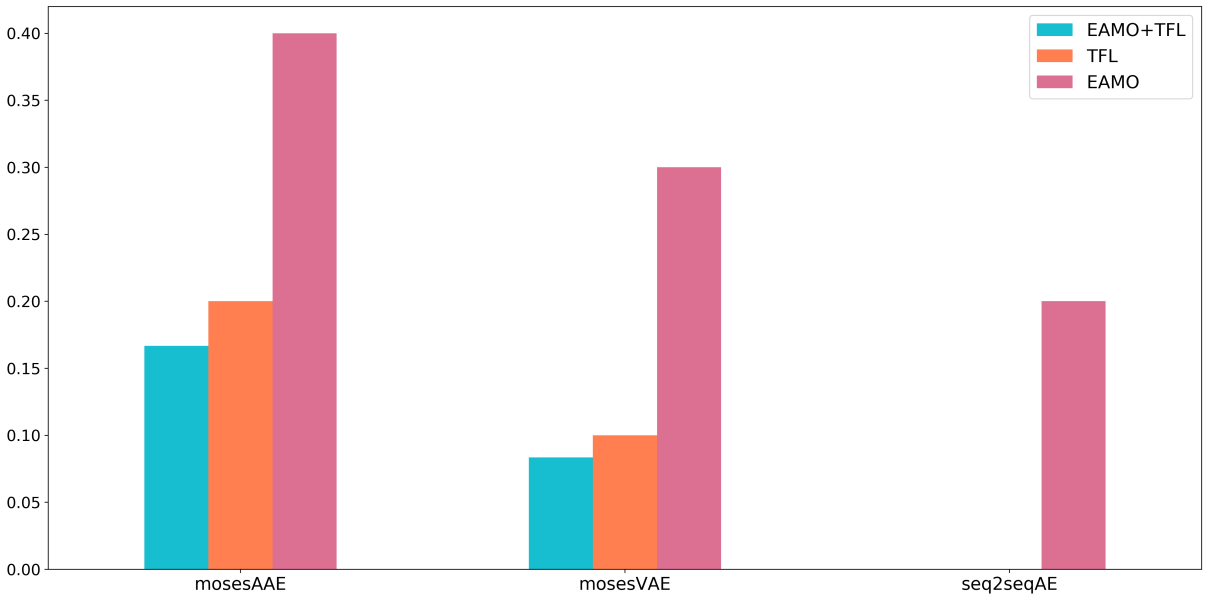


Figure 54: Acceptance rate of the various model/method combinations used for the first sweeteners set. The acceptance rate was used, instead of total counts, to allow direct comparisons as the total quantities being proposed varied depending on the specific model and method used.

- solubility (log) in the range]-2;4] ;
- ring sizes in the range [4;7] ;
- less than 5 stereoisomers;
- no polyphenols;
- no phenylethylamines;
- no sulphonamides.

These new screens directly resulted from the feedback received and, as with the previous ones, leveraged methods from the RDKit package. The matching of specific substructures, such as phenylethylamines, was implemented with SMARTS, also within RDKit. Figure 55 gives a visual illustration of the effect these filters would have had in the reviewed set of molecules. Simply applying the stereoisomer count filter would have identified 36 of the 73 rejections. Applying all the filters lead to the identification of 55 of the rejected molecules, over two thirds. These filters however, also result in some false positives removing 6 molecules that were otherwise accepted. From these, 3 contained sulphonamides, 2 were slightly above the stereoisomer count threshold and one was flagged for containing a substructure similar to phenylethylamine.

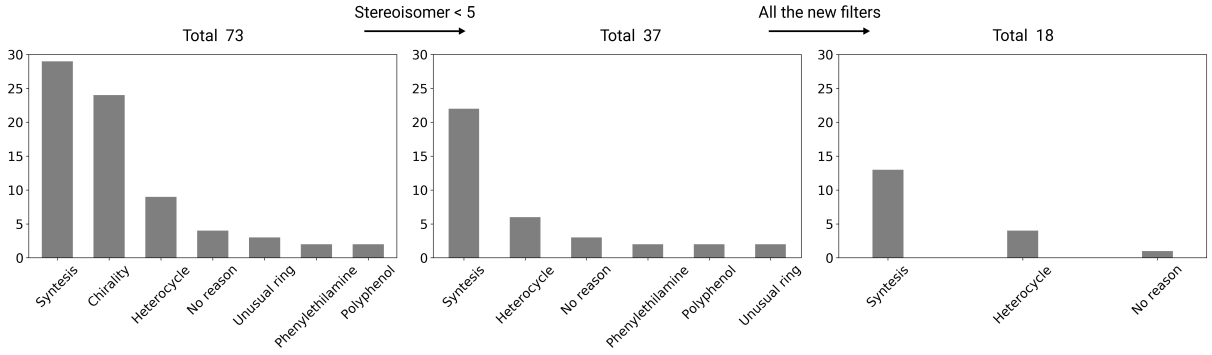


Figure 55: Effect in rejection causes among the first sweeteners set by applying the molecular quality filters. The leftmost plot shows the original counts, the middle one shows those counts after only applying the stereoisomer count filter and the rightmost plot shows those counts when all the filter are applied.

6.3.4 *Generation of the second set of putative sweeteners*

Following the receival of the previous feedback, we sought to improve our methods and generate a new set of sweet molecules. Similar to the previous workflow, both the prior models (before transfer learning) and the fine-tuned models were used within the EAMO framework to evolve molecules. For this new set however, both the prior and fine-tuned models were also directly sampled.

Another change implemented was in the EAMO method where, instead of combining every generation of a single population to obtain a suitable number of molecules, only the final generation was kept and the EA restarted several times. Furthermore, we allowed longer optimization runs of 500 generations to help the EA find better solutions. A new objective was also added to EAMO to help capture the quality of the molecules being evolved. This term was computed by the sum of 3 separate metrics, a score for fitting an acceptable solubility range measured as the logP; a penalty for rings with unusual sizes and a score for fitting a desirable molecular weight range. This new term was then optimized under the MOEA alongside the predicted sweetness, toxicity and natural product likeness.

For this new set, a total of around 100 molecules was also expected. The quotas of each model/method however, were adjusted following the previous results. Particularly, both the 'mosesVAE' and 'mosesAAE' were set to contribute with 10 molecules from each method. Meanwhile, the 'seq2seqAE' model was slated to only propose 5 compounds from each method except for EAMO with the prior model, the best performing method, which was kept on par with the other models. These adjustments reflect the results observed from the previous set, increasing the role played by 'mosesAAE' and decreasing the 'seq2seqAE' contributions.

The outline of the pipeline used for this second set of sweet molecules is illustrated in Figure 56. In addition to the filters used for the previous set, the new ones discussed above were also employed to help ensure the quality of the molecules being selected.

Table 14 shows the number of molecules from each model/method combination left after each step. Each model/method combination was initially set to generate around 10k molecules. However, after some preliminary tests it was concluded that, in order to meet the quotas, the 'mosesAAE' model would need

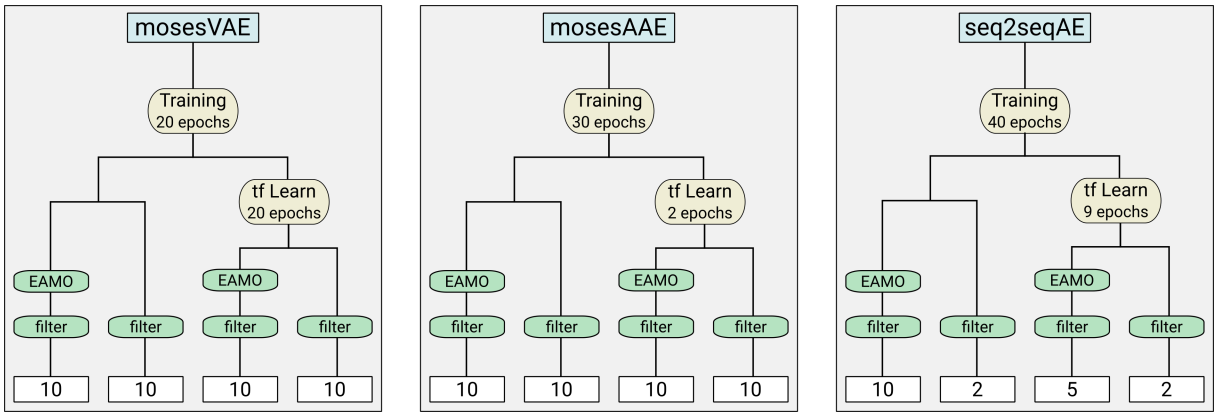


Figure 56: Pipeline used for second sweetener set. The three architectures were trained on 1.7M molecules from ZINC[29] and then fine-tuned on known sweeteners. The models bothe before and after fine-tuning where directly sampled and applied under EAMO. The generated molecules then passed a series of filtering steps resulting in a set o 99 new compounds.

to generate more molecules for all methods. Likewise, the 'seq2seqAE' model required that more initial compounds be generated when sampling both the prior and fine-tuned model. This was rectified by either increasing the number of repetitions in the EA or the number of random latent points sampled.

After being generated, molecules were then checked for validity, canonicalized and the duplicates removed. The filters from the first set were then applied, alongside the molecular quality filters discussed above. The similarity check then followed, however it was relaxed to 0.6 Tanimoto similarity between atom pair fingerprints. At the same time, it was also expanded to check against known sweeteners and molecules in the training dataset, besides checking against other generated molecules. As such, this step replaced the direct copies check performed for the last set. A final visual inspection confirmed that the molecules were reasonable and the final quantities were enforced, where necessary, through sampling.

Table 14: Molecules remaining after each step in the process of generating the second sweeteners set

Method	Model	Generated	Unique and Valid	Filters		Sim.	Final
				old	new		
Sample	mosesVAE	10240	9131	42	13	12	10
	mosesAAE	40960	36447	150	63	18	10
	seq2seqAE	40960	3715	13	2	2	2
EA	mosesVAE	10000	4135	529	121	117	10
	mosesAAE	30000	18533	60	23	11	10
	seq2seqAE	10000	3257	1112	231	213	10
TfL+EA	mosesVAE	10000	2769	652	233	186	10
	mosesAAE	30000	5345	355	80	50	10
	seq2seqAE	10000	4424	880	268	255	5
TfL	mosesVAE	10240	8672	48	17	16	10
	mosesAAE	30720	5496	135	16	14	10
	seq2seqAE	40960	4600	23	4	4	2

As the Table shows, even despite the adjustments to the initial quantities generated, the 'seq2seqAE' model still failed to meet all its quotas. In those instances the main cause for exclusion, besides the low validity rate, was due to molecules falling outside the MW range or having low predicted sweetness.

Comparing the stepwise rejection rates, or the percentage of molecules removed with each step, between Table 13 and Table 14 can allow a look into the impact of our improved methods. These rates were expected to remain unchanged when sampling the fine-tuned models, as neither the models or method were altered. However, a lower rate of valid and unique molecules was observed for the 'mosesAAE' model. This may be due to the increased sampling quantity which, coupled with its low uniqueness, may be approaching the limits of the models capacity. On the other hand, the new objective alongside longer optimization runs and a different sampling scheme should alter the rejection rates for the EAMO method. Indeed both 'mosesVAE' and 'seq2seqAE' saw improved results for validity and uniqueness as well as for the old filters. Meanwhile the 'mosesAAE' model performed worse in the first step and about equal for the old filters step, indicating that shorter optimization runs may have been sufficient for this architecture.

Within this new set, we also experimented with applying the new molecular quality filters before the old filters. In this instance a reduction of 50% was observed, meaning half of the molecules passed the new filters. This is in contrast with the higher 73% reduction experienced when the new filter were applied after the old ones. This seemingly suggests that the old filters, selecting for high predicted sweetness, non-toxicity and natural product likeness, also skew the selected molecules towards lower molecular quality. Meanwhile, thanks to its new molecular quality objective, it was expected that the EAMO method would fair better than the direct sampling at satisfying both the old and new filters. This however, was only observed with the fine-tuned models suggesting that the new objective could recover some of the molecular quality lost with transfer learning. At the same time, this also shows that there exists room to improve the new objective, as EAMO still produced lower molecular quality than directly sampling the prior models. A first step could be to include the substructure filtering criterion of the filters in EAMO, either as part of the molecular quality objective or as constraints.

Lastly, the final set containing 99 molecules was also sent to our partners for evaluation. No detailed feedback has been received at the time of writing, however, some preliminary comments point at the new compounds having higher molecular quality.

7

CONCLUSIONS

7.1 Summary of the Devised Work

This work set out to develop tools for designing novel molecules based on generative DL. We started with a review of the state-of-the art, gathering the most current approaches and methods being proposed. A special focus was placed on the methods being developed for the controlled generation of molecules as these were especially relevant for our intended case study of designing novel sweeteners. This review highlighted a number of promising architectures, from which 4 were selected for further consideration namely two VAEs, one AAE and a *seq2seq* based AE. From this review also emerged a number of possible approaches to the targeted generation of molecules. However, the use of EAs to navigate the latent space of chemical AEs seemed particularly appropriate given that EAs are a very versatile and effective optimization method and also have a long history of use for chemical design.

After reviewing other works, we then developed the DeepMolGen framework to support the implementation of various generative DL architectures, their training, evaluation and even transfer learning. This framework was then put to use with the inclusion of the 4 models gathered from the state-of-the-art which we could successfully train and evaluate. With one exception, a VAE, the models proved useful and capable of generating novel molecules.

Following that, our next step was then to combine the models from DeepMolGen with EAs to control the properties of the generated compounds. This resulted in the development of the EAMO framework which is model agnostic, can optimize for multiple molecular properties and leverage various evolutionary algorithms both single and multi objective. From the state-of-the-art review, a series of benchmarks were then compiled and used to evaluate EAMO, comparing its performance with other relevant methods. These case studies were completed with one of the aforementioned models and showed EAMO capable of controlling abstract chemical properties. Furthermore, it mostly outperformed or at least matched results from previous work, leading to the conclusion that EAMO is a viable tool for targeted molecular design.

Our last experiment dealt with the generation of promising novel sweeteners, a task which, to our knowledge, has not been attempted with generative DL. Furthermore, we had the special opportunity of having expert chemist evaluating and commenting on the generated molecules, providing an independent validation of the work being performed, as well as valuable input for improving our methods. For this, we leveraged 3 of the models mentioned above, the transfer learning capabilities of DeepMolGen and the

EAMO framework within a pipeline which produced a set of 102 novel molecules. These were then reviewed by our partners who provided valuable feedback. This then led to a second set of possible sweeteners being generated with improved methods. Comparing this set with the first highlighted several avenues through which further improvements should be possible. Nevertheless, preliminary feedback from our partners pointed at a significant improvement with the second set, allowing us to draw the conclusion that the task was completed with success.

Given these results, we conclude that our initial goals of developing generative DL-based tools for targeted molecular design were attained. Furthermore, we successfully reviewed recent literature relevant to this topic, explored various architectures and tested the developed methods, concluding in the design of novel sweeteners.

7.2 Main Contributions

The present work presented several contributions, that to the best of our knowledge are novel. Specifically:

- DeepMolGen, a framework standardizing the implementation, training and evaluation of state-of-the-art deep generative architectures;
- EAMO, a framework employing MOEAs to navigate the latent space of AE based models, allowing the targeted generation of molecules;
- A study into the combination of MOEAs and AEs to perform targeted molecular generation, comparing the results on various state-of-the-art case studies against other current methods;
  - This study, titled "Combining Multi-objective Evolutionary Algorithms with deep generative models towards focused molecular design" was accepted to the EvoApplications 24th European Conference on the Applications of Evolutionary and bio-inspired Computation, taking place in April 2021 in Seville, where the work will be orally presented.
- A systematic and critical review of the state-of-the-art on deep generative models, related optimization methods for targeted compound design, and their applications;
  - This study was submitted for the Journal of Chemical Information and Modeling, being at the time of writing under review a revised version submitted after a first version was considered to be publishable with “minor changes”.
- Generation of two sets of novel, non-toxic, and synthesizable promising sweeteners, totaling 208 new compounds;
  - These have been reviewed by expert chemists with a large portion synthesized and expected to undergo experimental validation.

7.3 Future Work

Having drawn these conclusion, it is now also worthwhile to appoint possible improvements and new avenues of research. For instance, besides SMILES, two other molecular representations have been implemented within DeepMolGen, namely SELFIES and DeepSMILES. These however, have not yet been tested or evaluated and, as such, their validation poses as an obvious next step.

Next, and due to the difficulties of evaluating generative models, the implementation of more metrics or even the inclusion of other benchmarking platforms such as Guacamole [84] would likely prove valuable and lead to increased confidence on the trained models.

Many of the early approaches to generating new molecules with DL borrowed from the NLP field, using RNNs for modeling and generating molecules as sequences. However, in the meantime, the state-of-the-art in language processing has evolved to leverage attention mechanisms within architectures like the transformer, BERT and GPT-3 [162, 163, 164]. Indeed, these advances have begun to trickle into the field of generative molecular design with recent works, leveraging the transformer architecture to approach the generation of new bioactive compounds as a translation from the amino acid sequence of a target protein to an active SMILES[165], to link fragments together[130] or to perform scaffold hopping [166]. As such, it would be useful to experiment with architectures based on transformers, or others from the literature, such as the Jin et al. [103] JT-VAE. This would both further validate DeepMolGen and provide other points of comparison in the benchmarks of Chapter 6.2 between EAMO and other methods. During the discussion of this last topic, a future improvement has already been highlighted relating to a possible improved implementation of the similarity constraint.

Regarding the generation of novel sweeteners, improvements have also already been discussed such as including the substructure filters as constraints in the EA, as well as improving on the molecular quality filters. These, along with improvements stemming from future feedback on the second set and the other additions detailed above could lead to the development of a new pipeline and the generation of more possible sweeteners with even higher quality.

Lastly, expanding beyond transfer learning and EAMO to other methods of targeted generation such as RL, BO or conditioned generation are enticing future directions, given that broad comparisons between different methods are missing from the current literature.

BIBLIOGRAPHY

[1] DiMasi, J.A., Grabowski, H.G., Hansen, R.W.: Innovation in the pharmaceutical industry: New estimates of R&D costs. Journal of Health Economics **47**, 20–33 (May 2016). https://doi.org/10.1016/j.jhealeco.2016.01.012

[2] Polishchuk, P.G., Madzhidov, T.I., Varnek, A.: Estimation of the size of drug-like chemical space based on GDB-17 data. J Comput Aided Mol Des p. 5 (2013)

[3] Schneider, G.: Automating drug discovery. Nature Reviews Drug Discovery **17**(2), 97–113 (Feb 2018). https://doi.org/10.1038/nrd.2017.232

[4] Schneider, P., Schneider, G.: De Novo Design at the Edge of Chaos: Miniperspective. Journal of Medicinal Chemistry **59**(9), 4077–4086 (May 2016). https://doi.org/10.1021/acs.jmedchem.5b01849

[5] Reymond, J.L.: The Chemical Space Project. Acc. Chem. Res. p. 9 (2015)

[6] Foster, D., Safari, a.O.M.C.: Generative deep learning: teaching machines to paint, write, compose, and play (2019)

[7] Segler, M.H., Kogej, T., Tyrchan, C., Waller, M.P.: Generating focused molecule libraries for drug discovery with recurrent neural networks. ACS central science **4**(1), 120–131 (2018)

[8] Merk, D., Friedrich, L., Grisoni, F., Schneider, G.: *De Novo* Design of Bioactive Small Molecules by Artificial Intelligence. Molecular Informatics **37**(1-2), 1700153 (Jan 2018). https://doi.org/10.1002/minf.201700153

[9] Merk, D., Grisoni, F., Friedrich, L., Schneider, G.: Tuning artificial intelligence on the de novo design of natural-product-inspired retinoid X receptor modulators. Communications Chemistry **1**(1), 68 (Dec 2018). https://doi.org/10.1038/s42004-018-0068-1

[10] Guimaraes, G.L., Sanchez-Lengeling, B., Outeiral, C., Farias, P.L.C., Aspuru-Guzik, A.: Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence Generation Models. arXiv:1705.10843 (May 2017)

[11] Blaschke, T., Olivecrona, M., et al.: Application of Generative Autoencoder in *De Novo* Molecular Design. Molecular Informatics **37**(1-2), 1700123 (Jan 2018). https://doi.org/10.1002/minf.201700123

[12] Winter, R., Montanari, F., Noé, F., Clevert, D.A.: Learning continuous and data-driven molecular descriptors by translating equivalent chemical representations. Chemical Science **10**(6), 1692–1701 (2019). https://doi.org/10.1039/C8SC04175J

[13] Sanchez-Lengeling, B., Aspuru-Guzik, A.: Inverse molecular design using machine learning: Generative models for matter engineering. Science **361**(6400), 360–365 (Jul 2018). https://doi.org/10.1126/science.aat2663

[14] Elton, D.C., Boukouvalas, Z., Fuge, M.D., Chung, P.W.: Deep learning for molecular design—a review of the state of the art. Molecular Systems Design & Engineering **4**(4), 828–849 (2019). https://doi.org/10.1039/C9ME00039A

[15] Schwalbe-Koda, D., Gómez-Bombarelli, R.: Generative Models for Automatic Chemical Design. arXiv preprint arXiv:1907.01632 (Jul 2019)

[16] Zhavoronkov, A., Vanhaelen, Q., Oprea, T.I.: Will artificial intelligence for drug discovery impact clinical pharmacology? Clinical Pharmacology & Therapeutics **107**(4), 780–785 (2020). https://doi.org/https://doi.org/10.1002/cpt.1795, https://ascpt.onlinelibrary.wiley.com/doi/abs/10.1002/cpt.1795

[17] Bian, Y., Xie, X.Q.: Generative chemistry: drug discovery with deep learning generative models. arXiv preprint arXiv:2008.09000 (2020)

[18] Engel, T., Gasteiger, J. (eds.): Chemoinformatics: basic concepts and methods. Wiley-VCH, Weinheim (2018), oCLC: 1012130305

[19] Faulon, J.L., Bender, A. (eds.): Handbook of chemoinformatics algorithms. Chapman & Hall/CRC mathematical and computational biology series, Chapman & Hall/CRC, Boca Raton, FL (2010), oCLC: ocn226357322

[20] Bunin, B.A. (ed.): Chemoinformatics: theory, practice, & products. Springer, Dordrecht (2007), oCLC: 77481986

[21] Leach, A.R., Gillet, V.J.: An introduction to chemoinformatics. Springer, Dordrecht, rev. ed edn. (2007), oCLC: 255664592

[22] Engel, T., Gasteiger, J. (eds.): Applied chemoinformatics: achievements and future opportunities. Wiley-VCH, Weinheim (2018), oCLC: 1034693178

[23] Weininger, D.: SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. Journal of Chemical Information and Modeling **28**(1), 31–36 (Feb 1988). https://doi.org/10.1021/ci00057a005

[24] Heller, S.R., McNaught, A., Pletnev, I., Stein, S., Tchekhovskoi, D.: InChI, the IUPAC International Chemical Identifier. Journal of Cheminformatics **7**(1), 23 (Dec 2015). https://doi.org/10.1186/s13321-015-0068-4

[25] Bajorath, J. (ed.): Chemoinformatics for Drug Discovery. John Wiley & Sons, Inc, Hoboken, NJ (2013). https://doi.org/10.1002/9781118742785.fmatter

[26] Wishart, D.S., Feunang, Y.D., Guo, A.C., Lo, E.J., Marcu, A., Grant, J.R., Sajed, T., Johnson, D., Li, C., Sayeeda, Z., Assempour, N., Iynkkaran, I., Liu, Y., Maciejewski, A., Gale, N., Wilson, A., Chin, L., Cummings, R., Le, D., Pon, A., Knox, C., Wilson, M.: DrugBank 5.0: a major update to the DrugBank database for 2018. Nucleic Acids Research **46**(D1), D1074–D1082 (Jan 2018). https://doi.org/10.1093/nar/gkx1037

[27] Gaulton, A., Bellis, L.J., Bento, A.P., Chambers, J., Davies, M., Hersey, A., Light, Y., McGlinchey, S., Michalovich, D., Al-Lazikani, B., Overington, J.P.: ChEMBL: a large-scale bioactivity database for drug discovery. Nucleic Acids Research **40**(D1), D1100–D1107 (Jan 2012). https://doi.org/10.1093/nar/gkr777

[28] Sun, J., Jeliazkova, N., Chupakhin, V., Golib-Dzib, J.F., Engkvist, O., Carlsson, L., Wegner, J., Ceulemans, H., Georgiev, I., Jeliazkov, V., Kochev, N., Ashby, T.J., Chen, H.: ExCAPE-DB: an integrated large scale dataset facilitating Big Data analysis in chemogenomics. Journal of Cheminformatics **9**(1), 17 (Dec 2017). https://doi.org/10.1186/s13321-017-0203-5

[29] Irwin, J.J., Sterling, T., Mysinger, M.M., Bolstad, E.S., Coleman, R.G.: ZINC: A Free Tool to Discover Chemistry for Biology. Journal of Chemical Information and Modeling **52**(7), 1757–1768 (Jul 2012). https://doi.org/10.1021/ci3001277

[30] Kim, S., Chen, J., Cheng, T., Gindulyte, A., He, J., He, S., Li, Q., Shoemaker, B.A., Thiessen, P.A., Yu, B., Zaslavsky, L., Zhang, J., Bolton, E.E.: PubChem 2019 update: improved access to chemical data. Nucleic Acids Research **47**(D1), D1102–D1109 (Jan 2019). https://doi.org/10.1093/nar/gky1033

[31] Ruddigkeit, L., van Deursen, R., Blum, L.C., Reymond, J.L.: Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17. Journal of Chemical Information and Modeling **52**(11), 2864–2875 (Nov 2012). https://doi.org/10.1021/ci300415d

[32] Shivanyuk, A., Ryabukhin, S., Tolmachev, A., Bogolyubsky, A., Mykytenko, D., Chupryna, A., Heilman, W., Kostyuk, A.: Enamine real database: Making chemical diversity real. Chemistry today **25**(6), 58–59 (2007)

[33] Huang, R., Xia, M., Nguyen, D.T., Zhao, T., Sakamuru, S., Zhao, J., Shahane, S.A., Rossoshek, A., Simeonov, A.: Tox21challenge to build predictive models of nuclear receptor and stress response pathways as mediated by exposure to environmental chemicals and drugs. Frontiers in Environmental Science **3**, 85 (2016)

[34] Ramakrishnan, R., Hartmann, M., Tapavicza, E., Von Lilienfeld, O.A.: Electronic spectra from tddft and machine learning in chemical space. The Journal of chemical physics **143**(8), 084111 (2015)

[35] Ramakrishnan, R., Dral, P.O., Rupp, M., Von Lilienfeld, O.A.: Quantum chemistry structures and properties of 134 kilo molecules. Scientific data **1**(1), 1–7 (2014)

[36] Wang, R., Fang, X., Lu, Y., Wang, S.: The pdbbind database: Collection of binding affinities for protein- ligand complexes with known three-dimensional structures. Journal of medicinal chemistry **47**(12), 2977–2980 (2004)

[37] Landrum, G.: Rdkit: open-source cheminformatics software (2016)

[38] Steinbeck, C., Hoppe, C., Kuhn, S., Floris, M., Guha, R., Willighagen, E.: Recent Developments of the Chemistry Development Kit (CDK) - An Open-Source Java Library for Chemo- and Bioinformatics. Current Pharmaceutical Design **12**(17), 2111–2120 (Jun 2006)

[39] O'Boyle, N.M., Banck, M., James, C.A., Morley, C., Vandermeersch, T., Hutchison, G.R.: Open Babel: An open chemical toolbox. Journal of Cheminformatics **3**(1), 33 (Dec 2011). https://doi.org/10.1186/1758-2946-3-33

[40] Konar, A.: Artificial intelligence and soft computing: behavioral and cognitive modeling of the human brain. CRC press (2018)

[41] Raschka, S.: Python machine learning: unlock deeper insights into machine learning with this vital guide to cutting-edge predictive analytics. Community experience distilled, Packt Publishing open source, Birmingham Mumbai (2016)

[42] Alpaydin, E.: Introduction to machine learning. Adaptive computation and machine learning, MIT Press, Cambridge, Mass, 2nd ed edn. (2010), oCLC: ocn317698631

[43] James, G., Witten, D., Hastie, T., Tibshirani, R. (eds.): An introduction to statistical learning: with applications in R. No. 103 in Springer texts in statistics, Springer, New York (2013)

[44] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://www.deeplearningbook.org

[45] Chollet, F.: Deep learning with Python. Manning Publications Co, Shelter Island, New York (2018)

[46] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/, software available from tensorflow.org

[47] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 1724–1734. Association for Computational Linguistics, Doha, Qatar (2014). https://doi.org/10.3115/v1/D14-1179

[48] Olah, C.: Understanding LSTM Networks (aug 2015), http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[49] Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., Bharath, A.A.: Generative adversarial networks: An overview. IEEE Signal Processing Magazine **35**(1), 53–65 (2018)

[50] Doersch, C.: Tutorial on variational autoencoders (2016)

[51] Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., Frey, B.: Adversarial autoencoders. arXiv preprint arXiv:1511.05644 (2015)

[52] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research **15**, 1929–1958 (jun 2014)

[53] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

[54] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d Alch e Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[55] Chollet, F., et al.: Keras. https://keras.io (2015)

[56] Walters, W.P.: Virtual Chemical Libraries: Miniperspective. Journal of Medicinal Chemistry **62**(3), 1116–1124 (Feb 2019). https://doi.org/10.1021/acs.jmedchem.8b01048

[57] Hartenfeller, M., Zettl, H., Walter, M., Rupp, M., Reisen, F., Proschak, E., Weggen, S., Stark, H., Schneider, G.: DOGS: reaction-driven de novo design of bioactive compounds. PLoS computational biology **8**(2), e1002380 (2012). https://doi.org/10.1371/journal.pcbi.1002380

[58] Varnek, A. (ed.): Tutorials in chemoinformatics. John Wiley & Sons, Inc, Hoboken, NJ (2017)

[59] Devi, R.V., Sathya, S.S., Coumar, M.S.: Evolutionary algorithms for de novo drug design – a survey. Applied Soft Computing **27**, 543 – 552 (2015). https://doi.org/https://doi.org/10.1016/j.asoc.2014.09.042

[60] Spiegel, J., Durrant, J.: Autogrow4: An open-source genetic algorithm for de novo drug design and lead optimization. Journal of Cheminformatics **12** (12 2020). https://doi.org/10.1186/s13321-020-00429-4

[61] Jensen, J.H.: 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). https://doi.org/10.1039/C8SC05372C

[62] Yoshikawa, N., Terayama, K., Sumita, M., Homma, T., Oono, K., Tsuda, K.: Population-based De Novo Molecule Generation, Using Grammatical Evolution. Chemistry Letters **47**(11), 1431–1434 (Nov 2018). https://doi.org/10.1246/cl.180665

[63] Olivecrona, M., Blaschke, T., Engkvist, O., Chen, H.: Molecular de-novo design through deep reinforcement learning. Journal of Cheminformatics **9**(1), 48 (Sep 2017). https://doi.org/10.1186/s13321-017-0235-x

[64] Gupta, A., Müller, A.T., Huisman, B.J.H., et al.: Generative Recurrent Networks for *De Novo* Drug Design. Molecular Informatics **37**(1-2), 1700111 (Jan 2018)

[65] van Deursen, R., Ertl, P., Tetko, I.V., Godin, G.: GEN: highly efficient SMILES explorer using autodidactic generative examination networks. Journal of Cheminformatics **12**(1), 22 (Apr 2020). https://doi.org/10.1186/s13321-020-00425-8

[66] Gómez-Bombarelli, R., Wei, J.N., Duvenaud, D., Hernández-Lobato, J.M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T.D., Adams, R.P., Aspuru-Guzik, A.: Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. ACS Central Science **4**(2), 268–276 (Feb 2018). https://doi.org/10.1021/acscentsci.7b00572

[67] Lim, J., Ryu, S., Kim, J.W., Kim, W.Y.: Molecular generative model based on conditional variational autoencoder for de novo molecular design. Journal of Cheminformatics **10**(1), 31 (Dec 2018). https://doi.org/10.1186/s13321-018-0286-7

[68] Yuan, W., Jiang, D., Nambiar, D.K., Liew, L.P., Hay, M.P., Bloomstein, J., Lu, P., Turner, B., Le, Q.T., Tibshirani, R., Khatri, P., Moloney, M.G., Koong, A.C.: Chemical Space Mimicry for Drug Discovery. Journal of Chemical Information and Modeling **57**(4), 875–882 (Apr 2017). https://doi.org/10.1021/acs.jcim.6b00754, publisher: American Chemical Society

[69] Polykovskiy, D., Zhebrak, A., Vetrov, D., Ivanenkov, Y., Aladinskiy, V., Mamoshina, P., Bozdaganyan, M., Aliper, A., Zhavoronkov, A., Kadurin, A.: Entangled Conditional Adversarial Autoencoder for de Novo Drug Discovery. Molecular Pharmaceutics **15**(10), 4398–4405 (Sep 2018). https://doi.org/10.1021/acs.molpharmaceut.8b00839

[70] O'Boyle, N., Dalke, A.: DeepSMILES: An Adaptation of SMILES for Use in Machine-Learning of Chemical Structures. preprint (Sep 2018). https://doi.org/10.26434/chemrxiv.7097960.v1

[71] Krenn, M., HHäse, F., Nigam, A., Friederich, P., Aspuru-Guzik, A.: Selfies: a robust representation of semantically constrained graphs with an example application in chemistry. arXiv preprint arXiv:1905.13741 (May 2019)

[72] Kusner, M.J., Paige, B., Hernández-Lobato, J.M.: Grammar variational autoencoder. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 1945–1954. JMLR. org (2017)

[73] Li, Y., Vinyals, O., Dyer, C., Pascanu, R., Battaglia, P.: Learning Deep Generative Models of Graphs. arXiv preprint arXiv:1803.03324 (Mar 2018)

[74] Liu, Q., Allamanis, M., Brockschmidt, M., Gaunt, A.: Constrained graph variational autoencoders for molecule design. In: Advances in Neural Information Processing Systems. pp. 7795–7804 (2018)

[75] Mercado, R., Rastemo, T., Lindelöf, E., Klambauer, G., Engkvist, O., Chen, H., Bjerrum, E.J.: Graph networks for molecular design. Machine Learning: Science and Technology **2**(2), 025023 (mar 2021). https://doi.org/10.1088/2632-2153/abcf91

[76] De Cao, N., Kipf, T.: Molgan: An implicit generative model for small molecular graphs. arXiv preprint arXiv:1805.11973 (2018)

[77] Simonovsky, M., Komodakis, N.: Graphvae: Towards generation of small graphs using variational autoencoders. In: International Conference on Artificial Neural Networks. pp. 412–422 (2018)

[78] Ma, T., Chen, J., Xiao, C.: Constrained Generation of Semantically Valid Graphs via Regularizing Variational Autoencoders. In: Advances in Neural Information Processing Systems. pp. 7113–7124 (2018)

[79] Hawkins, P.C.D.: Conformation generation: The state of the art. Journal of chemical information and modeling **57**(8), 1747–1756 (2017). https://doi.org/10.1021/acs.jcim.7b00221

[80] Skalic, M., Jiménez, J., Sabbadin, D., De Fabritiis, G.: Shape-based generative modeling for de novo drug design. Journal of Chemical Information and Modeling **59**(3), 1205–1214 (Mar 2019). https://doi.org/10.1021/acs.jcim.8b00706

[81] Gebauer, N., Gastegger, M., Schütt, K.T.: Symmetry-adapted generation of 3d point sets for the targeted discovery of molecules. In: NeurIPS (2019)

[82] Preuer, K., Renz, P., Unterthiner, T., Hochreiter, S., Klambauer, G.: Fréchet chemnet distance: A metric for generative models for molecules in drug discovery. Journal of chemical information and modeling **58**(9), 1736–1741 (2018). https://doi.org/10.1021/acs.jcim.8b00234

[83] Arús-Pous, J., Blaschke, T., Ulander, S., Reymond, J.L., Chen, H., Engkvist, O.: Exploring the gdb-13 chemical space using deep generative models. Journal of cheminformatics **11**(1), 1–14 (2019)

[84] Brown, N., Fiscato, M., Segler, M.H., Vaucher, A.C.: Guacamol: benchmarking models for de novo molecular design. Journal of chemical information and modeling **59**(3), 1096–1108 (2019)

[85] Polykovskiy, D., Zhebrak, A., Sanchez-Lengeling, B., Golovanov, S., Tatanov, O., Belyaev, S., Kurbanov, R., Artamonov, A., Aladinskiy, V., Veselov, M., et al.: Molecular sets (moses): a benchmarking platform for molecular generation models. Front. Pharmacol. **11** (2020)

[86] Renz, P., Van Rompaey, D., Wegner, J.K., Hochreiter, S., Klambauer, G.: On failure modes in molecule generation and optimization. Drug Discovery Today: Technologies **32-33**, 55–63 (2019). https://doi.org/doi.org/10.1016/j.ddtec.2020.09.003

[87] Cieplinski, T., Danel, T., Podlewska, S., Jastrzebski, S.: We should at least be able to design molecules that dock well. arXiv preprint arXiv:2006.16955 (2020)

[88] Zhang, J., Mercado, R., Engkvist, O., Chen, H.: Comparative study of deep generative models on chemical space coverage. ChemRxiv (2020). https://doi.org/10.26434/chemrxiv.13234289.v3

[89] Moret, M., Friedrich, L., Grisoni, F., Merk, D., Schneider, G.: Generative molecular design in low data regimes. Nature Machine Intelligence **2**(3), 171–180 (Mar 2020). https://doi.org/10.1038/s42256-020-0160-y

[90] Blaschke, T., Arús-Pous, J., Chen, H., Margreitter, C., Tyrchan, C., Engkvist, O., Papadopoulos, K., Patronov, A.: REINVENT 2.0 – an AI Tool for De Novo Drug Design. preprint (Apr 2020). https://doi.org/10.26434/chemrxiv.12058026.v1

[91] Bung, N., Krishnan, S.R., Bulusu, G., Roy, A.: De novo design of new chemical entities (nces) for sars-cov-2 using artificial intelligence (Mar 2020). https://doi.org/10.26434/chemrxiv.11998347.v1

[92] Li, Y., Zhang, L., Liu, Z.: Multi-objective de novo drug design with conditional graph generative model. Journal of Cheminformatics **10**(1), 33 (Dec 2018). https://doi.org/10.1186/s13321-018-0287-6

[93] Blaschke, T., Engkvist, O., Bajorath, J., Chen, H.: Memory-assisted reinforcement learning for diverse molecular de novo design (Jul 2020). https://doi.org/10.26434/CHEMRXIV.12693152.V1

[94] Zhavoronkov, A., Ivanenkov, Y.A., Aliper, A., Veselov, M.S., Aladinskiy, V.A., Aladinskaya, A.V., Terentiev, V.A., Polykovskiy, D.A., Kuznetsov, M.D., Asadulaev, A., Volkov, Y., Zholus, A., Shayakhmetov, R.R., Zhebrak, A., Minaeva, L.I., Zagribelnyy, B.A., Lee, L.H., Soll, R., Madge, D., Xing, L., Guo, T.,

Aspuru-Guzik, A.: Deep learning enables rapid identification of potent DDR1 kinase inhibitors. Nature Biotechnology **37**(9), 1038–1040 (Sep 2019). https://doi.org/10.1038/s41587-019-0224-x

[95] Popova, M., Shvets, M., Oliva, J., Isayev, O.: MolecularRNN: Generating realistic molecular graphs with optimized properties. arXiv:1905.13372 [cs, q-bio, stat] (May 2019), arXiv: 1905.13372

[96] Sanchez-Lengeling, B., Outeiral, C., Guimaraes, G.L., Aspuru-Guzik, A.: Optimizing distributions over molecular space. an objective-reinforced generative adversarial network for inverse-design chemistry (organic). ChemRxiv p. 530968 (2017)

[97] Putin, E., Asadulaev, A., Vanhaelen, Q., Ivanenkov, Y., Aladinskaya, A.V., Aliper, A., Zhavoronkov, A.: Adversarial Threshold Neural Computer for Molecular *de Novo* Design. Molecular Pharmaceutics **15**(10), 4386–4397 (Oct 2018). https://doi.org/10.1021/acs.molpharmaceut.7b01137

[98] Putin, E., Asadulaev, A., Ivanenkov, Y., Aladinskiy, V., Sanchez-Lengeling, B., Aspuru-Guzik, A., Zhavoronkov, A.: Reinforced Adversarial Neural Computer for *de Novo* Molecular Design. Journal of Chemical Information and Modeling **58**(6), 1194–1204 (Jun 2018). https://doi.org/10.1021/acs.jcim.7b00690

[99] You, J., Liu, B., Ying, Z., Pande, V., Leskovec, J.: Graph convolutional policy network for goal-directed molecular graph generation. In: Advances in neural information processing systems 31. pp. 6410–6421 (2018)

[100] Karimi, M., Hasanzadeh, A., Shen, Y.: Network-principled deep generative models for designing drug combinations as graph sets. Bioinformatics **36**(Supplement_1), i445–i454 (2020)

[101] Griffiths, R.R., Hernández-Lobato, J.M.: Constrained Bayesian optimization for automatic chemical design using variational autoencoders. Chemical Science **11**(2), 577–586 (2020). https://doi.org/10.1039/C9SC04026A

[102] Dai, H., Tian, Y., Dai, B., Skiena, S., Song, L.: Syntax-directed variational autoencoder for structured data. arXiv preprint arXiv:1802.08786 (2018)

[103] Jin, W., Barzilay, R., Jaakkola, T.: Junction Tree Variational Autoencoder for Molecular Graph Generation. arXiv preprint arXiv:1802.04364 (Mar 2019), arXiv: 1802.04364

[104] Samanta, B., De, A., Jana, G., Chattaraj, P.K., Ganguly, N., Rodriguez, M.G.: NeVAE: A Deep Generative Model for Molecular Graphs. Proceedings of the AAAI Conference on Artificial Intelligence **33**, 1110–1117 (Jul 2019). https://doi.org/10.1609/aaai.v33i01.33011110

[105] Bresson, X., Laurent, T.: A Two-Step Graph Convolutional Decoder for Molecule Generation. arXiv:1906.03412 [cs, stat] (Jun 2019), arXiv: 1906.03412

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

[107] Prykhodko, O., Johansson, S.V., Kotsias, P.C., Arús-Pous, J., Bjerrum, E.J., Engkvist, O., Chen, H.: A de novo molecular generation method using latent vector based generative adversarial network. Journal of Cheminformatics **11**(1), 74 (2019)

[108] Maziarka, Ł., Pocha, A., Kaczmarczyk, J., Rataj, K., Danel, T., Warchoł, M.: Mol-cyclegan: a generative model for molecular optimization. Journal of Cheminformatics **12**(1), 1–18 (2020)

[109] Sattarov, B., Baskin, I.I., Horvath, D., Marcou, G., Bjerrum, E.J., Varnek, A.: De novo molecular design by combining deep deep autoencoder recurrent neural networks with generative topographic mapping. Journal of Chemical Information and Modeling **59**(3), 1182–1196 (mar 2019). https://doi.org/10.1021/acs.jcim.8b00751

[110] Winter, R., Montanari, F., Steffen, A., Briem, H., Noé, F., Clevert, D.A.: Efficient multi-objective molecular optimization in a continuous latent space. Chemical science **10**(34), 8016–8024 (2019)

[111] Chenthamarakshan, V., Das, P., Hoffman, S.C., Strobelt, H., Padhi, I., Lim, K.W., Hoover, B., Manica, M., Born, J., Laino, T., Mojsilovic, A.: CogMol: Target-Specific and Selective Drug Design for COVID-19 Using Deep Generative Models. arXiv:2004.01215 [cs, q-bio, stat] (Jun 2020), arXiv: 2004.01215

[112] Kotsias, P.C., Arús-Pous, J., Chen, H., Engkvist, O., Tyrchan, C., Bjerrum, E.J.: Direct steering of de novo molecular generation with descriptor conditional recurrent neural networks. Nature Machine Intelligence **2**(5), 254–265 (2020)

[113] Shayakhmetov, R., Kuznetsov, M., Zhebrak, A., Kadurin, A., Nikolenko, S., Aliper, A., Polykovskiy, D.: Molecular Generation for Desired Transcriptome Changes With Adversarial Autoencoders. Frontiers in Pharmacology **11**, 269 (apr 2020). https://doi.org/10.3389/fphar.2020.00269

[114] Méndez-Lucio, O., Baillif, B., Clevert, D.A., Rouquié, D., Wichard, J.: De novo generation of hit-like molecules from gene expression signatures using artificial intelligence. Nature Communications **11**(1), 1–10 (2020)

[115] Born, J., Manica, M., Oskooei, A., Cadow, J., Borgwardt, K., Martínez, M.R.: Paccmannrl: Designing anticancer drugs from transcriptomic data via reinforcement learning. arXiv preprint arXiv:1909.05114 (2019)

[116] Jin, W., Yang, K., Barzilay, R., Jaakkola, T.: Learning multimodal graph-to-graph translation for molecular optimization. arXiv:1812.01070 [cs, stat] (Jan 2019), arXiv: 1812.01070

[117] Masuda, T., Ragoza, M., Koes, D.R.: Generating 3d molecular structures conditional on a receptor binding site with deep generative models. arXiv preprint arXiv:2010.14442 (2020)

[118] Kang, S., Cho, K.: Conditional Molecular Design with Deep Generative Models. Journal of Chemical Information and Modeling **59**(1), 43–52 (Jan 2019). https://doi.org/10.1021/acs.jcim.8b00263

[119] Lim, J., Hwang, S.Y., Moon, S., Kim, S., Kim, W.Y.: Scaffold-based molecular design with a graph generative model. Chemical Science **11**(4), 1153–1164 (2020). https://doi.org/10.1039/C9SC04503A

[120] Kadurin, A., Aliper, A., Kazennov, A., Mamoshina, P., Vanhaelen, Q., Khrabrov, K., Zhavoronkov, A.: The cornucopia of meaningful leads: Applying deep adversarial autoencoders for new molecule development in oncology. Oncotarget **8**(7), 10883–10890 (Feb 2017). https://doi.org/10.18632/oncotarget.14073

[121] Frazier, P.I.: A tutorial on bayesian optimization. arXiv:1807.02811 [cs, math, stat] (Jul 2018), arXiv: 1807.02811

[122] Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., De Freitas, N.: Taking the human out of the loop: A review of bayesian optimization. Proceedings of the IEEE **104**(1), 148–175 (2015)

[123] Das, P., Sercu, T., Wadhawan, K., Padhi, I., Gehrmann, S., Cipcigan, F., Chenthamarakshan, V., Strobelt, H., Santos, C.d., Chen, P.Y., Yang, Y.Y., Tan, J., Hedrick, J., Crain, J., Mojsilovic, A.: Accelerating Antimicrobial Discovery with Controllable Deep Generative Models and Molecular Dynamics. arXiv:2005.11248 [cs, q-bio, stat] (May 2020), arXiv: 2005.11248

[124] Kingma, D.P., Mohamed, S., Rezende, D.J., Welling, M.: Semi-supervised learning with deep generative models. In: Advances in neural information processing systems. pp. 3581–3589 (2014)

[125] Gao, W., Coley, C.W.: The synthesizability of molecules proposed by generative models. Journal of chemical information and modeling **60**(12), 5714–5723 (2020)

[126] Horwood, J., Noutahi, E.: Molecular design in synthetically accessible chemical space via deep reinforcement learning. ACS Omega **5**(51), 32984–32994 (2020). https://doi.org/10.1021/acsomega.0c04153

[127] Gottipati, S.K., Sattarov, B., Niu, S., Pathak, Y., Wei, H., Liu, S., Blackburn, S., Thomas, K., Coley, C., Tang, J., et al.: Learning to navigate the synthetically accessible chemical space using reinforcement learning. In: International Conference on Machine Learning. pp. 3668–3679. PMLR (2020)

[128] Bradshaw, J., Paige, B., Kusner, M.J., Segler, M., Hernández-Lobato, J.M.: Barking up the right tree: an approach to search over molecule synthesis dags. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 6852–6866. Curran Associates, Inc. (2020)

[129] Imrie, F., Bradley, A.R., van der Schaar, M., Deane, C.M.: Deep generative models for 3d linker design. Journal of chemical information and modeling **60**(4), 1983–1995 (2020)

[130] Yang, Y., Zheng, S., Su, S., Zhao, C., Xu, J., Chen, H.: Syntalinker: automatic fragment linking with deep conditional transformer neural networks. Chemical Science **11**, 8312–8322 (2020). https://doi.org/10.1039/D0SC03126G

[131] Tan, X., Jiang, X., He, Y., Zhong, F., Li, X., Xiong, Z., Li, Z., Liu, X., Cui, C., Zhao, Q., Xie, Y., Yang, F., Wu, C., Shen, J., Zheng, M., Wang, Z., Jiang, H.: Automated design and optimization of multitarget schizophrenia drug candidates by deep learning. European Journal of Medicinal Chemistry **204**, 112572 (2020). https://doi.org/https://doi.org/10.1016/j.ejmech.2020.112572

[132] Yang, Y., Zhang, R., Li, Z., Mei, L., Wan, S., Ding, H., Chen, Z., Xing, J., Feng, H., Han, J., Jiang, H., Zheng, M., Luo, C., Zhou, B.: Discovery of highly potent, selective, and orally efficacious p300/cbp histone acetyltransferases inhibitors. Journal of medicinal chemistry **63**(3), 1337–1360 (2020). https://doi.org/10.1021/acs.jmedchem.9b01721

[133] Grisoni, F., Huisman, B., Button, A., Moret, M., Atz, K., Merk, D., Schneider, G.: Combining generative artificial intelligence and on-chip synthesis for de novo drug design (Dec 2020). https://doi.org/10.26434/chemrxiv.13498587.v1

[134] Shaker, N., Abou-Zleikha, M., AlAmri, M., Mehellou, Y.: A generative deep learning approach for the discovery of sars cov2 protease inhibitors (apr 2020). https://doi.org/10.26434/chemrxiv.12170337.v1

[135] Born, J., Manica, M., Cadow, J., Markert, G., Mill, N.A., Filipavicius, M., Martínez, M.R.: Paccmannrl on sars-cov-2: Designing antiviral candidates with conditional generative models. CoRR (2020)

[136] Hachmann, J., Olivares-Amaya, R., Atahan-Evrenk, S., Amador-Bedolla, C., Sánchez-Carrera, R.S., Gold-Parker, A., Vogt, L., Brockway, A.M., Aspuru-Guzik, A.: The harvard clean energy project: large-scale computational screening and design of organic photovoltaics on the world community grid. The Journal of Physical Chemistry Letters **2**(17), 2241–2251 (2011)

[137] Jørgensen, P.B., Mesta, M., Shil, S., Lastra, J.M.G., Wedel, K., Thygesen, K.S., Schmidt, M.N.: Machine learning-based screening of complex molecules for polymer solar cells. J. Chem. Phys. p. 14 (2018)

[138] Yuan, Q., Santana-Bonilla, A., Zwijnenburg, M.A., Jelfs, K.E.: Molecular generation targeting desired electronic properties via deep generative models. Nanoscale **12**(12), 6744–6758 (2020)

[139] Ramsundar, B., Eastman, P., Walters, P., Pande, V., Leswing, K., Wu, Z.: Deep Learning for the Life Sciences. O'Reilly Media (2019), https://www.amazon.com/Deep-Learning-Life-Sciences-Microscopy/dp/1492039837

[140] Deb, K., Jain, H.: An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. IEEE Transactions on Evolutionary Computation **18**(4), 577–601 (2014). https://doi.org/10.1109/TEVC.2013.2281535

[141] Zitzler, E., Laumanns, M., Thiele, L.: Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. vol. 3242 (01 2001)

[142] Kukkonen, S., Lampinen, J.: Gde3: the third evolution step of generalized differential evolution. In: 2005 IEEE Congress on Evolutionary Computation. vol. 1, pp. 443–450 Vol.1 (2005). https://doi.org/10.1109/CEC.2005.1554717

[143] Tonda, A.: Inspyred: Bio-inspired algorithms in python. Genetic Programming and Evolvable Machines pp. 1–4 (2019)

[144] Benitez-Hidalgo, A., Nebro, A.J., Garcia-Nieto, J., Oregi, I., Del Ser, J.: jmetalpy: A python framework for multi-objective optimization with metaheuristics. Swarm and Evolutionary Computation **51**, 100598 (2019)

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

[146] Bickerton, G.R., Paolini, G.V., Besnard, J., Muresan, S., Hopkins, A.L.: Quantifying the chemical beauty of drugs. Nature chemistry **4**(2), 90–98 (2012). https://doi.org/10.1038/nchem.1243

[147] Ertl, P., Schuffenhauer, A.: Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. Journal of cheminformatics **1**(1), 1–11 (2009)

[148] Ertl, P., Roggo, S., Schuffenhauer, A.: Natural product-likeness score and its application for prioritization of compound libraries. Journal of chemical information and modeling **48**(1), 68–74 (2008)

[149] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014)

[150] Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. arXiv preprint arXiv:1409.3215 (2014)

[151] Rogers, D., Hahn, M.: Extended-connectivity fingerprints. Journal of Chemical Information and Modeling **50**(5), 742–754 (2010). https://doi.org/10.1021/ci100050t, https://doi.org/10.1021/ci100050t

[152] Ravber, M., Mernik, M., Črepinšek, M.: The impact of quality indicators on the rating of multi-objective evolutionary algorithms. Applied Soft Computing **55**, 265 – 275 (2017)

[153] Tuwani, R., Wadhwa, S., Bagler, G.: Bittersweet: Building machine learning models for predicting the bitter and sweet taste of small molecules. Scientific reports **9**(1), 1–13 (2019)

[154] Dagan-Wiener, A., Nissim, I., Abu, N.B., Borgonovo, G., Bassoli, A., Niv, M.Y.: Bitter or not? bitterpredict, a tool for predicting taste from chemical structure. Scientific reports **7**(1), 1–13 (2017)

[155] Banerjee, P., Preissner, R.: Bittersweetforest: a random forest based binary classifier to predict bitterness and sweetness of chemical compounds. Frontiers in chemistry **6**, 93 (2018)

[156] Rojas, C., Todeschini, R., Ballabio, D., Mauri, A., Consonni, V., Tripaldi, P., Grisoni, F.: A qstr-based expert system to predict sweetness of molecules. Frontiers in chemistry **5**, 53 (2017)

[157] Zhong, M., Chong, Y., Nie, X., Yan, A., Yuan, Q.: Prediction of sweetness by multilinear regression analysis and support vector machine. Journal of food science **78**(9), S1445–S1450 (2013)

[158] Ojha, P.K., Roy, K.: Development of a robust and validated 2d-qspr model for sweetness potency of diverse functional organic molecules. Food and Chemical Toxicology **112**, 551–562 (2018)

[159] Chéron, J.B., Casciuc, I., Golebiowski, J., Antonczak, S., Fiorucci, S.: Sweetness prediction of natural compounds. Food chemistry **221**, 1421–1425 (2017)

[160] Garg, N., Sethupathy, A., Tuwani, R., Nk, R., Dokania, S., Iyer, A., Gupta, A., Agrawal, S., Singh, N., Shukla, S., et al.: Flavordb: a database of flavor molecules. Nucleic acids research **46**(D1), D1210–D1216 (2018)

[161] Belitz, H.D., Grosch, W., Schieberle, P.: Food cemistry (2009)

[162] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. arXiv preprint arXiv:1706.03762 (2017)

[163] Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)

[164] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. arXiv preprint arXiv:2005.14165 (2020)

[165] Grechishnikova, D.: Transformer neural network for protein-specific de novo drug generation as a machine translation problem. Sci. Rep. **11**(1), 1–13 (2021)

[166] Zheng, S., Lei, Z., Ai, H., Chen, H., Deng, D., Yang, Y.: Deep scaffold hopping with multi-modal transformer neural networks (2020)