Machine Learning Methods for Modeling Synthesizable Molecules

John Anthony Bradshaw

Department of Engineering
University of Cambridge

This thesis is submitted for the degree of
Doctor of Philosophy

Trinity Hall

February 2021

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the preface and specified in the text. It is not substantially the same as any work that has been previously submitted for any degree or other qualification except as declared in the preface and specified in the text. This thesis does not exceed the prescribed word limit for the Engineering Degree Committee. More specifically, it contains fewer than 65,000 words including appendices, footnotes, tables, and equations and has fewer than 150 figures.

John Anthony Bradshaw
February 2021

Machine Learning Methods for Modeling Synthesizable Molecules

John Anthony Bradshaw

The search for new molecules often involves cycles of design-make-test-analyze steps, where new molecules are designed, synthesized in a lab, tested, and then analyzed to inform what is to be designed next. This thesis proposes new machine learning (ML) methods to augment chemists in the design and make steps of this process, focusing on the tasks of (a) how to use ML to predict chemical reaction outcomes, and (b) how to build generative models to search for new molecules. We take a common approach to both tasks, building our ML models around existing powerful tools and abstractions from the field of chemistry, and in doing so, show that the tasks we tackle are intrinsically linked.

Reaction prediction is important for validating synthesis plans before carrying them out. Many previous ML approaches to reaction prediction have treated reactions as either a black box translation or a single graph edit operation. Instead, we propose a model (ELECTRO) that predicts the reaction products through modeling a sequence of electron movements. We show how modeling electron movements in this way has the benefit of being easy for chemists to interpret, and also is a natural format in which to incorporate the constraints of chemistry, such as balanced atom counts before and after a reaction. We show that our model achieves excellent performance on an important subset of chemical reactions and recovers a basic knowledge of chemistry without explicit supervision.

In designing new models to search for molecules with particular properties, it is important that the models describe not only what molecule to make, but also crucially how to make it. These instructions form a synthesis plan, describing how easy-to-obtain building blocks can be combined together to form more complex molecules of interest through chemical reactions. Inspired by this real-world process, we develop two machine learning approaches that incorporate reactions into the virtual generation of new molecules. We show that aligning our model with the real-world process allows us to better link up the design and make steps involved in molecule search, and permits chemists to examine the practicability of both the final molecules we suggest and their synthetic routes. Molecule search is inherently an extrapolation task, and we show that by building our methods around the inductive biases of modeling reactions, we can generalize to new chemical spaces, suggesting molecules that not only perform well, but are synthesizable too.

Acknowledgements

I am very grateful to my supervisors, José Miguel Hernández-Lobato, Zoubin Ghahramani, and Bernhard Schölkopf. Miguel made this thesis possible with his amazing ability to advise at every level, from getting models working to suggesting new research ideas. Zoubin convinced me to start the PhD and throughout it has always given sage counsel. Bernhard has established an excellent research environment, and I appreciate the freedom he offered me to explore it.

I also owe a huge debt of thanks to Brooks Paige, Marwin Segler, and Matt Kusner. The work presented in this thesis has been done in collaboration with them and Miguel. Their contributions in developing new ideas, bringing clarity to the process, and devising interesting experiments cannot be overstated. All four have taught me an enormous amount, from how one should approach and solve problems to how one should communicate and present results. I am extremely grateful for the interactions we have had and hope that long may they continue!

I am grateful to other collaborators during my studies. Alex Matthews offered direction in my initial research projects and first introduced me to the idea of applying machine learning to molecules. Thank you to Yingzhen Li and Yash Sharma for the great collaboration, and the insightful conversations with Yingzhen about adversarial examples and generative modeling.

I would also like to thank the large number of people who improved my time in Cambridge and Tübingen, including Alex Neitz, Austin Tripp, Brian Trippe, Chaochao Lu, Diego Agudelo-España, Giambattista Parascandolo, Gregor Simm, Jiri Hron, Maria Lomeli, Matej Balog, Mateo Rojas-Carulla, Matthias Bauer, Paul Rubenstein, and all the other members of the CBL and MPI groups. I am grateful to my Cambridge-Tübingen program colleagues, in particular my cohort: Alessandro Ialongo, who contributed to perhaps the greatest discovery of my PhD, the "Tenison Road Sandwich", and Niki Kilbertus, who has always been ready to "chew the fat". Many thanks to the administrators who ensured all ran smoothly, especially Catherine Munn, Diane Hazell, Rachel Fogg, and Sabrina Rehbaum. In addition, I wish to thank James Hensman and his team for hosting me on an enjoyable internship. I am also grateful to Razvan Ranca and Ken Chatfield for introducing me to a series of interesting applied problems; I greatly enjoyed our brainstorming sessions followed by fish & chips!

This thesis exists thanks to financial support from the Engineering and Physical Sciences Research Council and the Max Planck Society, as well as computational resources from The Alan Turing Institute and open source software tools and libraries, all of which I greatly appreciate.

Finally, many thanks to my family and friends – in particular Julia, who with her clear-headedness, kind-heartedness, and quick-wittedness brings joy to my life, and my parents and brother for their unconditional love and support, which made getting here all possible.

Table of Contents

Chapter 1

Introduction

The search for new molecules can have an enormous positive benefit on society. Perhaps no where is this more true than in the field of drug discovery. Lichtenberg (2005) found that between 1986 and 2000, the launches of new drugs accounted for 40% of the increase in life expectancy achieved over that period. Unfortunately, drug discovery is also a hugely expensive process. Here, the search for new molecules contributes to the huge overall costs needed to bring a new drug to market, estimated to be roughly one to two billion US dollars (DiMasi et al., 2016; Paul et al., 2010). To put this into perspective, this is the same order of magnitude as the yearly healthcare expenditure of Iceland in 2010 ($\approx$ 1.2 billion US dollars; The World Bank Group, 2020). Worryingly, these costs also appear to be rising as research and development productivity falls (Bunnage, 2011; Kola and Landis, 2004; Pammolli et al., 2011; Scannell et al., 2012). Increased costs can exacerbate problems already faced in funding the search for treatments for neglected tropical diseases or orphan diseases, which already lack a profitable market (Hunt, 2007; Ioset and Chang, 2011, p. 1361).

A high proportion of these costs come down to the high number of design-make-test-analyze cycles required in searching for new molecules. In each iteration of this cycle a chemist designs a new molecule, which then gets made and tested, the results of which drive what gets designed next. There is a strong incentive to build ML (machine learning) methods to augment chemists in these tasks, either by reducing the overall number of cycles required or by reducing the time taken in performing the individual steps of each cycle. But in building these methods we should be mindful of the complex interactions involved between chemist, computer, and chemicals. These inform what we require from our methods. We want our methods to generalize robustly to new domains and provide interpretable results. We want our methods to communicate in understandable formats, taking advantage of the powerful abstractions of chemistry built up over centuries of scientific endeavor. Most importantly though, we want to make sure we design our

models to answer the correct question; often we do not want to know just what happens, but why it happens, or even how to make it happen.

This thesis attempts to build such machine learning methods. In particular we focus on building tools for the design and make steps of the molecular search process, making these steps more efficient, but also in the process bringing the two of them closer together. When *making* molecules, computational models for predicting the results of chemical reactions are useful for validating synthesis plans or providing insight into how and why reactions occur. In this thesis we design such a model for predicting the outcomes of chemical reactions, built around the abstraction of modeling electron movements in the reactant molecules. When *designing* molecules, the molecules suggested need to be synthesizable, or in other words they should be able to be made using these aforementioned chemical reactions. In the second half of this thesis, we therefore propose new generative models of molecules built around chemical reactions. We show that by building these physically inspired notions into our models as inductive biases, our models can often generalize to new chemical spaces successfully and make interpretable predictions.

1.1 Overview and main contributions

In this section we provide an overview of the structure of this thesis and detail the contributions we have made. The rest of this thesis consists of five more chapters; these are broken down as follows:

Chapter 2, Representing Molecules The next chapter introduces the background required for the later chapters. We introduce the concept of being able to model molecules using a series of different representations. This will form a key theme of the work in the rest of the thesis, where we model not just single molecules but also their interactions. We also briefly describe the recent progress in graph neural networks, which will form an essential component of the models we later present.

Chapter 3, Reaction Prediction This chapter focuses on the task of predicting the outcomes of chemical reactions. Reactions can be described as the stepwise redistribution of electrons in molecules. In turn for a large class of chemical reactions, this redistribution can be described by a single electron path through the molecules. In this chapter, we develop a model called ELECTRO that directly predicts this electron path. Predicting the electron path has the advantages of (a) being easy for chemists to interpret, (b) naturally encoding the sparsity of reactions (where only a few atoms or bonds change), and (c) incorporating the constraints of chemistry

(such as conservation of atom counts). We develop a method for extracting approximate electron paths, to train and test ELECTRO on, from large reaction datasets, and show how ELECTRO compared favorably to the state-of-the-art approaches at the time. Furthermore, we demonstrate how ELECTRO recovers a basic knowledge of chemistry without explicitly being trained to do so.

This chapter corresponds with the work previously published in:

> John Bradshaw, Matt J. Kusner, Brooks Paige, Marwin H. S. Segler, José Miguel Hernández-Lobato. "A Generative Model For Electron Paths". *International Conference on Learning Representations (ICLR)*. 2019. (Bradshaw et al., 2019a)
> (Code: `https://github.com/john-bradshaw/electro`)

Chapter 4, Searching for Synthesizable Molecules The fourth chapter focuses on the topic of molecule search. When suggesting a new molecule for a particular task, it is not only important *what* to make, but also crucially *how* to make it. These instructions form what is known as a synthesis plan, in effect a recipe for a particular molecule, describing the manner in which it is constructed from simpler reactants through chemical reactions. While there has been impressive recent progress in developing deep generative models of molecules (representing molecules as strings, trees, and even graphs), these have, on the whole, ignored synthesizability. Therefore, in this chapter we propose a new model, MOLECULE CHEF, which reflects a more realistic real-world process of creating a molecule through a reaction. More specifically, our generative model first proposes a bag of initial reactants (selected from a pool of commercially-available building block molecules) and then uses a reaction model to predict how they react together to generate a new molecule. We show that MOLECULE CHEF can generate a wide range of valid, unique, and novel molecules. Compared to previous generative approaches, our method generates synthetic routes, giving chemists an additional output on which to interrogate the viability of the molecules produced. Furthermore, we also demonstrate how a generative model of molecules like ours can be used on new tasks, such as retrosynthesis.

This chapter corresponds with the work previously published in:

> John Bradshaw, Brooks Paige, Matt J. Kusner, Marwin H. S. Segler, José Miguel Hernández-Lobato. "A Model to Search for Synthesizable Molecules" *Advances in Neural Information Processing Systems (NeurIPS)*. 2019. (Bradshaw et al., 2019b)
> (Code: `https://github.com/john-bradshaw/molecule-chef`)

Chapter 5, Barking up the Right Tree: Generating Molecule Synthesis DAGs The fifth chapter extends the ideas from Chapter 4. Chapter 4 drew attention to the fact that the search for molecules with particular properties is inherently a constrained optimization problem: molecules found must be synthesizable. However, our solution to this in Chapter 4, MOLECULE CHEF, was designed around only a single reaction, and as such, we were overly restrictive in our constraints. Many common molecules actually require several reaction steps to obtain. Therefore, in Chapter 5 we extend the ideas of MOLECULE CHEF to multiple reactions. We describe how one can represent complicated synthesis plans as directed acyclic graphs (DAGs), and develop a generative model, which we call DOG-GEN, over this structure. We show how one can use DOG-GEN as a component in larger ML architectures and frameworks, such as (a) latent variable models (for sampling and interpolation), as well as (b) reinforcement learning–based optimization routines (to find molecules with particular properties). We demonstrate that DOG-GEN is able to search over the same space that a chemist would have access to, obtaining competitive scores with a series of powerful unconstrained baselines, while suggesting molecules that are synthesizable and stable too.

This chapter corresponds with the work previously published in:

> John Bradshaw, Brooks Paige, Matt J. Kusner, Marwin H. S. Segler, José Miguel Hernández-Lobato. "Barking up the right tree: an approach to search over molecule synthesis DAGs". *Advances in Neural Information Processing Systems (NeurIPS)*. 2020. (Bradshaw et al., 2020)
> (Code: `https://github.com/john-bradshaw/synthesis-dags`)

Chapter 6, Conclusions and Future Directions The final chapter summarizes the content of the previous chapters, suggests future directions for the work, and reiterates our main theme: that to build robust and interpretable machine learning models for chemistry, we should develop methods around sensible inductive biases, taking advantage of the abstractions and tools already built up over centuries of development.

As indicated in the references above, the work in this thesis has been done in collaboration with Brooks Paige, Matt Kusner, Marwin H.S. Segler, and José Miguel Hernández-Lobato.

Note to reader The structure of the chapters broadly follows the original publications, although parts have been rearranged and the discussion supplemented in places. Chapters 4 and 5 are closely related, and to avoid a repeated discussion of related work, Section 4.1

acts as a background for both of them. In the main text, we try to avoid including overly specific details on the hyperparameters and architectures used, but provide these in the appendices for those interested. Finally, at points we also make use of boxes, such as the one below:

Box 1.1: This is a box

These provide clarification and further context for topics introduced in the main text. They can be safely skipped by readers already familiar with the concepts involved.

Chapter 2

Representing Molecules

In this chapter we describe different ways molecules can be represented. More specifically, we describe different approaches for how feature vectors of molecules (molecule-level "representations" or "embeddings") can either be designed or learned, particularly with a focus on how these can then be used as inputs for our ML (machine learning) models. We break this description down, starting with older chemoinformatics approaches for representing molecules, before then going on to explain recent graph neural network (GNN) methods for computing molecule-level representations. Finally, we include a brief discussion of extensions to 3D representations and the strengths and weaknesses of such an approach.

Understanding molecular representation plays a crucial role in the chapters that follow. In particular, the GNN models we discuss will form a core component of the later architectures that we develop. Also, although we only discuss representations of molecules as *input* to ML models in this chapter, this discussion also helps introduce concepts that will be important when we come to develop models that can generate molecules as *output* in Chapters 4 and 5. Finally, even if we do not directly use the 3D representations, discussed in the final part of this chapter, we believe that this topic is important to briefly discuss so that we can be mindful of the limitations of current approaches, and of opportunities to improve them. In providing such a background, we assume the reader is already familiar with the basics of modern deep learning (Goodfellow et al., 2016, § II). More detailed reviews into the chemistry-related aspects of this chapter can be found in Brown (2009) or Gasteiger and Engel (2003).

2.1 What is a molecule?

What is a molecule? A molecule is a collection of charged particles, negative electrons and positive nuclei, existing at particular locations in 3D space (Jensen, 2017, p. 1)[1]. A pertinent next question to ask, and the topic of this chapter, is how can we then represent such an entity *to a computer*? There is not a unique answer, but actually several possible ways to represent a molecule, depending on our modeling assumptions and the abstractions we make. For example, Figure 2.1 shows a selection of representations for the molecule paracetamol (also known as acetaminophen); from left to right we have (i) the molecular graph representation, (ii) the adjacency matrix/node feature list representation, and (iii) finally the character-based SMILES and InChI representations. We shall go through each of these in more detail below.

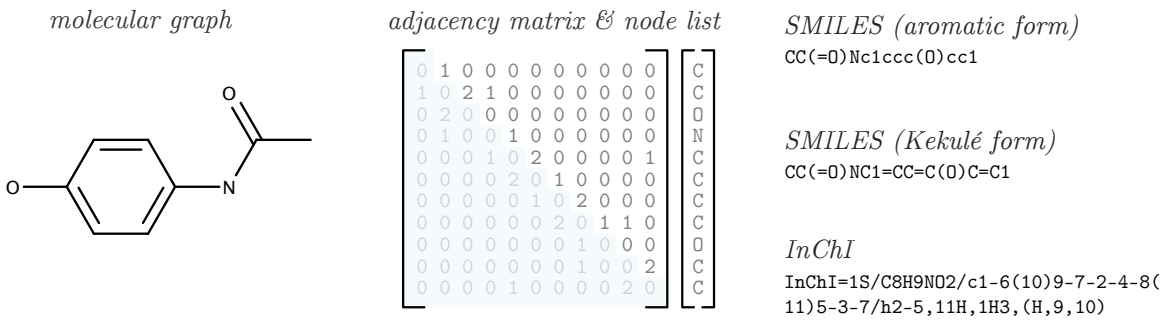


Figure 2.1 *Different possible representations of paracetamol (also known as acetaminophen). Displayed from left to right are (i) the molecular graph representation (in Kekulé form), (ii) the adjacency matrix and node feature list representation, and (iii) the SMILES (Weininger, 1988) and InChI (Heller et al., 2015) text representations. The molecular graph, when drawn in this style, is also referred to as a skeletal formula, and has the convention that carbon atoms are not generally labeled. Note, that due to its symmetry, the bottom half of the adjacency matrix, shaded here in blue, is redundant and can be reconstructed from its upper-half.*

2.1.1 Graph representation

On the left of Figure 2.1 is the molecular graph representation. Formally, we can represent a graph by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with a set of nodes (also called vertices) $\mathcal{V}$ and a set of edges $\mathcal{E}$. In this notation, a directed edge existing from node $u \in \mathcal{V}$ to node $v \in \mathcal{V}$ is denoted as an ordered pair, $(u, v) \in \mathcal{E}$. In a *molecular* graph, for molecule $\mathcal{M}$, the nodes of the graph

[1] To be more precise, a molecule is defined as "an electrically neutral entity consisting of more than one atom" (IUPAC, 2014). However, we shall not make a distinction between molecules, ions, or even single atoms here and will model and treat them identically.

represent the atoms, while the edges represent the covalent bonds between these atoms. This means that we can associate with each node, $v \in \mathcal{V}$, an attribute vector $\mathbf{x}_v \in \mathbb{R}^{d'}$, describing the node's atom type, charge, and any other atom-specific details. Likewise, with each edge we associate the single attribute $e_{(u,v)} \in \Delta$, encoding its bond type[2]. In the molecular graphs we consider there is no direction associated with each bond, which is accounted for in the directed graph formalism by an identical directed edge in each direction.

There are two further aspects about the molecular graph worth pointing out. Firstly, hydrogen atoms are often excluded; if required the number of these attached to each atom can be inferred from an atom's type, charge, and degree (Sayle, 2013). Secondly, sometimes it is convenient to manipulate or display (as in Figure 2.1) molecules in their Kekulé form (as opposed to their aromatic form; Gasteiger, 2003, pp. 37-39). In the Kekulé form, the aromatic parts of molecules are represented by alternating single and double bonds. There are several possible ways to convert molecules in the aromatic form to a Kekulé form, and this process is called kekulization (Trinajstic, 2018, Chap. 8). Kekulization can also be done in a standardized (i.e. deterministic) way to choose a unique Kekulé form from the ones available (Hähnke et al., 2018, Fig. 35), which is useful if two molecules need to be compared. An advantage of working with the Kekulé form is that it simplifies the number of possible edge attributes required; in the Kekulé form there is no aromatic bond and each bond must consist of an integer number of electron pairs: 1 pair, 2 pairs, and 3 pairs for single, double, and triple bonds respectively.

2.1.2 Adjacency matrices

When it comes to storing and manipulating molecular graphs in a computer, it is sometimes easier to work with the adjacency matrix, **Adj**, and node feature list, **V**, shown in the middle of Figure 2.1. The adjacency matrix details whether a bond exists between two atoms at indices i and j, $\mathbf{Adj}_{ij} > 0$, and if so, its type, encoded as a real number. Due to the undirected nature of these bonds, the adjacency matrix for molecular graphs will always be symmetric. The node feature list is a stacked representation of the node attributes, describing the atom type associated with each node. Both of these matrices are not unique for a molecule; for instance we can permute the rows and columns of the adjacency matrix

[2]We can represent single, aromatic, double, and triple bonds using members of the alphabet $\Delta = \{1, 1.5, 2, 3\}$, such that the edge attribute describes how many pairs of electrons make up the bond. Although higher order bond types can also exist, for instance between two metals (Radius and Breher, 2006), we will not be interested in them here.

(using an identical permutation for both the rows and the columns) and still represent the same molecule.

2.1.3 String representations

As a more condensed encoding, text-based representations of molecules are often favored; they offer the associated advantages of usually being faster for people to read and write. For instance, we can refer to common molecules by one of their traditional names: "paracetamol" or "acetaminophen" for the molecule shown in Figure 2.1. However, these traditional names have only been defined for certain molecules and not every potential new molecule we could come across has been given such a name. Therefore, instead, we could use a molecule's systematic IUPAC[3] name (Favre and Powell, 2014), for instance "N-(4-hydroxyphenyl)acetamide" for paracetamol, but these quickly become cumbersome for larger molecules!

Line notations The systematic name is an example of a wider concept referred to as a line notation representation, in which molecules are described as a linear sequence of letters, numbers, and other special characters, groups of which explicitly describe a sub-part of the molecular structure (Gasteiger and Engel, 2003, § 2.3; Wiswesser, 1985). In effect, while it is futile to give traditional names to all possible *entire* molecules, we can break these molecules down into simpler parts to create a more manageable vocabulary. Then to describe a new molecule, we can compose together these simpler sub-parts to describe a much more complex whole, in a similar way to which we can compose words together in natural languages to form sentences.

There exist a variety of different types of line notation, each often introduced to fulfill a particular need. For instance, WLN (Wiswesser Line Notation; Wiswesser, 1982), introduced in the 1940s, and where paracetamol takes the representation "QR DMV1", was designed to describe the increasingly large number of molecules being discovered during that period. It then later gained particular popularity in the 1960s and 1970s for its compact notation, during a time when computer file storage was at a premium (Gasteiger, 2003, § 1.5; Wiswesser, 1985). Subsequent notations, such as SLN (SYBYL Line Notation; Ash et al., 1997), enabled humans to *input* complex substructure queries into computers, such that they could interact with (and take advantage of) the more advanced computer databases that had been developed, and the new kinds of queries these systems enabled. More recently, this relationship between humans, notation, and

[3] International Union of Pure and Applied Chemistry.

computer has been somewhat flipped. New notations such as SELFIES (Krenn et al., 2020) or DeepSMILES (O'Boyle and Dalke, 2018) have been designed around being amenable for a ML algorithm to *output*, a topic which we shall come back to in Chapters 4 and 5. Two currently popular line notations that we shall describe in more detail below are the SMILES (Weininger, 1988) and InChI (Heller et al., 2015) notations, shown in the right-hand part of Figure 2.1.

SMILES The SMILES notation (simplified molecular-input line-entry system; Weininger, 1988), shown at the top right of Figure 2.1, represents a traversal over the graph's nodes and edges. Atoms met on this traversal are represented by their atomic symbols; single, double, triple, quadruple, and aromatic bonds by the symbols "-", "=", "#", "$", and ":" respectively; branches by parenthesis; and cycles by matching integers. Single bonds are also often inferred implicitly, so the symbol "." is used to denote the separation between two different molecules.

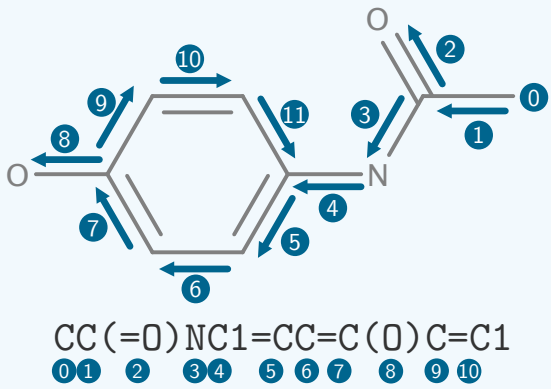
SMILES has different advantages and disadvantages compared to other line notations. An advantage of SMILES, apart from its widespread implementation and adoption, is its semi-readable nature. An issue with SMILES is that it is not unique; there are several possible representations for each molecule depending on how the traversal through the atoms in the graph is done (see Box 2.1).

InChI The issue present in SMILES of having many possible strings for the same molecule is avoided in the InChI representation (IUPAC International Chemical Identifier), which finds a unique notation for each molecule (Heller et al., 2015, pp. 25-29; McKay, 1981). A key feature of InChIs are their layered structure, with later, optional layers refining the finer details of the molecule. As an example, in the InChI shown in Figure 2.1, there are four layers: (i) the prefix layer, which is independent of the molecule and describes the version of the InChI encoding used[4]; (ii) the empirical formula layer; (iii) a layer describing the skeletal connections; and finally (iv) a layer that describes the hydrogens. A shorter 27-character representation of an InChI can also be obtained by a special hash function; this representation is called an InChIKey and is particularly useful with search engines.

InChIs have obtained widespread use (Heller et al., 2013; Warr, 2015). However, as they are arguably less human-readable than SMILES strings, they have not supplanted these latter representations, and SMILES representations remain popular for many tasks. This has been helped by the development of different "canonicalization" algorithms that

[4]As such this is often not counted as an "official" layer (Heller et al., 2015, pp. 5-7).

Box 2.1: Canonical SMILES



The SMILES string is formed by performing a depth-first search through a molecule (see figure to the left) starting at a head atom (marked 0 in the figure) (Weininger, 1988; Weininger et al., 1989). This means that there are multiple possible SMILES strings for a given molecular graph. For instance, for this same paracetamol molecule any of `CC(=O)NC1=CC=C(O)C=C1`, `C1=CC(NC(=O)C)=CC=C1O`, or `C1=C(NC(C)=O)C=CC(O)=C1` are valid SMILES strings. These different variants come about from two types of decisions we can make when performing the depth-first search, namely (a) which atom (i.e. node) should we start at, and (b) which neighboring atom should we pick next when there are multiple choices?

We can form unique or "canonical" SMILES strings by making sure we pick the same atom, regardless of their initial numbering, when faced with either of these decisions. On a high level, the way that this is often done is to first assign a permutation-invariant *score* to each of the atoms. Then when facing a decision about which atom to consider next in the search, this can be resolved by choosing the one with the lower score.

There are many different ways one could choose to generate the initial, permutation-invariant score; examples include using (a) the Morgan algorithm (Morgan, 1965), which actually predates the SMILES representation, (b) the CANON algorithm initially proposed for SMILES by Weininger et al. (1989), or (c) even more recent approaches such as those described in O'Boyle (2012); Schneider et al. (2015b). While we will not go into the details of how these work here, the basic idea is usually the same. One starts with assigning an initial permutation-invariant score to each node, such as its degree, and then one sequentially updates these scores by performing permutation-invariant operations, such as sums, on the scores of the neighbors. In this manner these algorithms share many similarities with the fingerprint and graph neural network methods described in Section 2.2 of this chapter. The process of choosing a canonical ordering with which to traverse molecules also arises as a topic of interest in ML methods for generating graphs (for example see Chapters 4 and 5, or Liao et al., 2019, § 2.3).

ensure a "canonical" (i.e. unique) SMILES string for each molecule (see Box 2.1), including techniques that use ideas from the InChI (O'Boyle, 2012).

2.2 Molecules as vectors

We will often want to form molecule-level embeddings (also called representations), $\boldsymbol{m}_{\mathcal{M}} \in \mathbb{R}^d$. These embeddings can then be used as feature vectors for representing molecules in a variety of ML models, whether generic, application-agnostic regression or classification models (such as ordinary feedforward neural networks), or more chemistry-specific models, such as the ones we shall introduce in the later chapters.

These embeddings can be computed in different ways. A few approaches directly take advantage of some of the representations we saw in the previous sections. For example, Goh et al. (2017) develop a convolutional neural network (CNN) model directly on pixelated images of the molecular graph. Alternatively, others have proposed starting from the SMILES string representation, and encoding this into one-hot vectors suitable as input for recurrent neural networks (RNNs; see for example the work of Bjerrum, 2017; Gómez-Bombarelli et al., 2018; Jastrzębski et al., 2016; or the sequence-to-sequence methods described in the next chapter).

However, none of the just-mentioned methods take into account the permutation invariance of their inputs. For example, we could rotate images of the molecular graph around by any number of degrees to create what looks like a new input to our model, even though the underlying molecule is unchanged. This can be somewhat rectified by training on augmented datasets (Goh et al., 2018); however, perhaps a more compelling approach is the use of *permutation-invariant* embedding methods. Two such methods we shall describe next are (i) 2D molecular fingerprints and (ii) graph neural networks (GNNs).

2.2.1 Two dimensional (2D) molecular fingerprints

Molecular fingerprints are vectors that can be used to represent a molecule. Often these fingerprints are binary vectors, where each entry (i.e. bit) of this vector describes the presence or absence of one or more substructures.[5] Fingerprints can be split into two

[5]The 2D molecular fingerprints we describe here are only one class of a more general family of traditional descriptors from the field of chemoinformatics. In practice, often descriptors from different families are concatenated together to form molecule-level embeddings. For example, globally-computed features, such as the molecular weight or the octanol-water partition coefficient (logP; Wildman and Crippen, 1999), might be appended to purely substructure-count–based measures, such as those we discuss. We refer the reader to Todeschini and Consonni (2000) for an extensive list of possible descriptors.

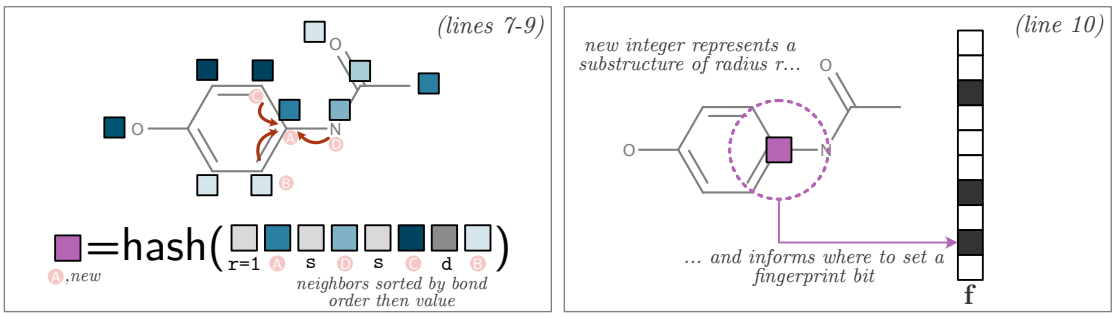


Figure 2.2 *Cartoon demonstrating some salient features of extended connectivity fingerprints (ECFPs; see Alg. 2.1), namely that at each step the integer associated with each atom (shown by the colored boxes) is updated using its neighbors, which in turn means it represents larger and larger substructures. Note, that while this figure captures the general principle of forming an extended connectivity fingerprint, we have omitted some often implemented subtleties, such as the deduplication of equivalent substructure bits (see Rogers and Hahn, 2010, p. 746).*

Algorithm 2.1: *Extended connectivity fingerprints (ECFPs; Rogers and Hahn, 2010)*

Input:

- Molecular graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with associated atom attributes $\mathbf{x}_v$ and edge attributes $e_{(u,v)}$;
- Hash function, $\mathsf{hash}(\cdot)$, that maps an array to an integer;
- Radius, R.

Result:

- Binary fingerprint vector $\boldsymbol{f}$.

```
f ← 0                                          ▷ Binary fingerprint (FP) is initialized as all zeros.
for v in V do
    h_{v,0} ← hash(x_v).                       ▷ Hash atom attributes to a single integer for each atom.
    f_{h_{v,0}} ← 1                            ▷ Set FP bit at index given by hash function.
for r in [1, ..., R] do
    for v in V do
        n ← [(e_{(u,v)}, h_{u,r-1}) for u in N(v)]   ▷ Form an array of information from the neighbors.
        m ← [r, h_{v,r-1}] ‖ sorted(n)         ▷ Concatenate neighbor information with information from atom.
        h_{v,r} ← hash(m)                      ▷ Hash array back down to a single integer.
        f_{h_{v,r}} ← 1                        ▷ Set FP bit at index given by hash function.
return f
```

groups based on how they associate different substructures with each bit: (a) fragment-dictionary fingerprints and (b) hashed fingerprints (Leach and Gillet, 2007, pp. 62-64). In fragment-dictionary fingerprints, for example MACCS (Durant et al., 2002), a mapping (i.e. dictionary) is maintained between a substructure and which bit(s) it should set. This means that given a fingerprint vector describing a specific molecule and the dictionary used in its construction, one can reconstruct the possible substructures present.

A disadvantage of these fragment-dictionary fingerprints is that domain knowledge is required when constructing the dictionary. Ideally, different molecules should map to distinct fingerprints, which means that you want to choose substructures for the keys of your dictionary that occur at relatively equal frequencies and somewhat independently of one another. The second type of fingerprint, hashed fingerprints, avoids this problem by not requiring a set of defined substructures upfront. Instead, a hash function is used to map any substructure one might encounter to an associated location in the embedding vector. An example of such a fingerprint in this family is the group of extended connectivity fingerprints (ECFPs; Rogers and Hahn, 2010). Figure 2.2 provides an overview of how these work.

Fingerprints were initially designed for substructure matching (Leach and Gillet, 2007, p. 101). Here they can be used for screening out possible matches between molecules, by first ensuring that the molecules' fingerprint vectors match, before more expensive checks are undertaken (Engel and Gasteiger, 2018, pp. 239-240). In this manner, when using hashed fingerprints, they can be seen as similar to the probabilistic data structure called a Bloom filter (Bloom, 1970), with the potential for false positive matches from hash collisions[6] but no false negatives. Fingerprints have also found applications in many other areas, such as diversity analysis or drug side effect prediction (Rogers and Hahn, 2010, Table 1). More recently, ECFPs and other kinds of 2D fingerprints have also found success as the input features for deep learning methods (Dahl et al., 2014; Ma et al., 2015; Ramsundar et al., 2015; Unterthiner et al., 2014; Wu et al., 2018).

2.2.2 Graph neural networks

As opposed to fixed fingerprints, we can use graph neural networks (GNNs). These can be seen as a parameterized, relaxed, and differentiable version of a more traditional substructure fingerprint, such as those previously discussed (Duvenaud et al., 2015, Fig. 2).

[6] A bit could have been set by different substructures in the different molecules due to the non-injective nature of the hash functions used. Although it is important to note that due to fingerprints' binary nature, and in particular the fact that fingerprints do not take account of the number of times a single substructure occurs, fingerprint matching may provide false positives for other reasons.

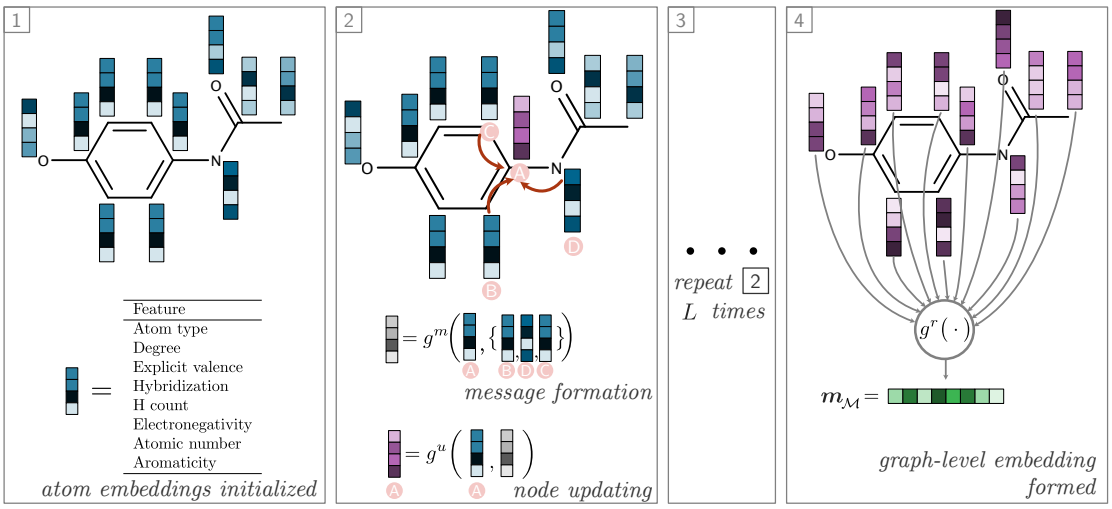


Figure 2.3 *Cartoon demonstrating how a graph neural network (GNN) operates on a molecular graph to form node-level and/or graph-level embeddings (see Alg. 2.2): (1) nodes are initialized with features derived from their respective atoms' properties; (2-3) several rounds of message passing steps take place, updating the node embeddings, using information from the neighbors; (4) graph-level embeddings are produced using a "readout" function. There is some flexibility in the choice of the particular functional forms used in the GNN (see Box 2.2); a key idea here is that the functions should be permutation invariant to set-type arguments, such as a node's neighbors, which can be achieved through functions such as sum, max, average, etc.*

Algorithm 2.2: *Graph neural network (GNN)*

Input:

- Molecular graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with associated node attributes $\mathbf{x}_v$ and edge attributes $e_{(u,v)}$;
- Parameterized functions: $g^m(\cdot,\cdot)$, $g^u(\cdot,\cdot)$, and $g^r(\cdot)$ (see § A.1 for possible functional forms);
- Number of propagation steps, L.

Result:

- Final node embeddings, $\{\boldsymbol{h}_{v,L}, \forall v \in \mathcal{V}\}$;
- Graph-level embedding, $\boldsymbol{m}_{\mathcal{M}}$.

/* Initialization: */

for v *in* $\mathcal{V}$ **do**

  $\boldsymbol{h}_{v,0} \leftarrow \mathbf{x}_v$ ▷ The node embeddings are initialized with the atom features.

/* Part 1, message passing: */

for l *in* $[1, \ldots, L]$ **do**

  for v *in* $\mathcal{V}$ **do**

    $\boldsymbol{m}_{v,l} \leftarrow g^m\left(\boldsymbol{h}_{v,l-1}, \{(\boldsymbol{h}_{w,l-1}, e_{(w,v)}), \forall w \in \mathcal{N}(v)\}\right)$ ▷ Form a message for the node from its neighbors.

    $\boldsymbol{h}_{v,l} \leftarrow g^u\left(\boldsymbol{h}_{v,l-1}, \boldsymbol{m}_{v,l}\right)$ ▷ We update the node using its previous embedding and the message.

/* Part 2, readout: */

$\boldsymbol{m}_{\mathcal{M}} \leftarrow g^r\left(\{\boldsymbol{h}_{v,L}, \forall v \in \mathcal{V}\}\right)$ ▷ Readout function maps set of all node embeddings to a graph-level embedding.

return $\{\boldsymbol{h}_{v,L}, \forall v \in \mathcal{V}\}, \boldsymbol{m}_{\mathcal{M}}$

Similar to ECFPs (extended connectivity fingerprints), graph neural networks look at increasingly larger substructures around atoms in a series of iterative steps. However, unlike ECFPs these networks can learn which substructures and characteristics are important when forming molecule-level embeddings. Furthermore, GNNs also allow for smoother relationships between similar molecules' embeddings through their continuous, i.e. no longer binary, nature.

Graph neural networks can be broken down into two parts, an initial message passing part, followed by an optional, final readout part, as shown in Algorithm 2.2 and Figure 2.3 (Gilmer et al., 2017). The message passing part can be further broken down into a series of L layers, akin to a regular feedforward neural network. Each of these layers iteratively refine the node-level embeddings (i.e. atom-level embeddings when dealing with molecular graphs), taking into account information (the "messages") from neighboring nodes. We will denote these node-level embeddings for node v, produced from the l-th layer, as $\boldsymbol{h}_{v,l}$, with the initial node embeddings set using the atom attributes, introduced in the previous section, i.e. $\boldsymbol{h}_{v,0} = \mathbf{x}_v$.

Each layer of message passing can be further broken down into two stages, (i) message formation and (ii) node updating. These stages involve two parameterized functions, $g^m(\cdot,\cdot)$ and $g^u(\cdot,\cdot)$, the exact structure of which we shall come back to later. In the first stage a "message" for each node in the graph, $\boldsymbol{m}_{v,l}$, is computed by taking account of information from the node's neighbors:

$$\boldsymbol{m}_{v,l} = g^m\left(\boldsymbol{h}_{v,l-1}, \left\{\left(\boldsymbol{h}_{w,l-1}, e_{(w,v)}\right), \forall\, w \in \mathcal{N}(v)\right\}\right), \tag{2.1}$$

where $\mathcal{N}(v)$ denotes the set of neighbors of node v in the graph. These messages are then used, along with the previous node embedding, in the second stage of the layer, to *update* the node's embedding:

$$\boldsymbol{h}_{v,l} = g^u\left(\boldsymbol{h}_{v,l-1}, \boldsymbol{m}_{v,l}\right). \tag{2.2}$$

The final node embeddings, $\boldsymbol{h}_{v,L}$, can be used as input to machine learning models for node-level tasks. Here it is often convenient to stack them as the rows of a matrix, $\boldsymbol{H}_{\mathcal{M}}$. Alternatively, they can be fed in as input to a further parameterized "readout" function (Gilmer et al., 2017, § 2), also sometimes called an aggregation function (Johnson, 2017, § 3), $g^r(\cdot)$, to produce a graph-level embedding (i.e. molecule-level embedding):

$$\boldsymbol{m}_{\mathcal{M}} = g^r\left(\left\{\boldsymbol{h}_{v,L}, \forall\, v \in \mathcal{V}\right\}\right). \tag{2.3}$$

There is some flexibility in choosing the precise functional forms of the functions $g^m(\cdot,\cdot)$, $g^u(\cdot,\cdot)$, and $g^r(\cdot)$, leading to a proliferation of approaches. A short discussion of the various approaches is given in Box 2.2[7]. The key idea in all of them is to maintain invariance to the specific ordering of arguments that take the form of a set. This is often done by using a sum, as addition is a commutative operation, although other permutation-invariant operations such as taking the average or the maximum of the set can also be used (Bloem-Reddy and Teh, 2020; Zaheer et al., 2017). The particular variant of graph neural network that we shall use in the experiments in the next few chapters will be Gated Graph Sequence Neural Networks (GGNNs), proposed by Li et al. (2015). We will use an attention-weighted sum readout function, as it has been found to work better than simple sums (Li et al., 2018a, § B.1). We provide further details about this architecture in Appendix A.1.

Box 2.2: Graph neural networks: a plethora of options

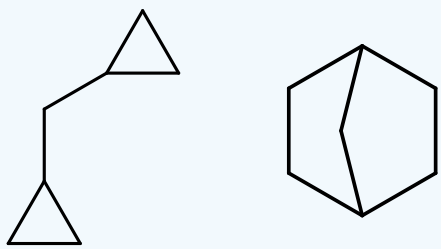
Graph neural networks (GNNs) have existed in various forms for several decades (Gori et al., 2005; Kireev, 1995; Merkwirth and Lengauer, 2005; Scarselli et al., 2009). For example, Kireev (1995) proposed ChemNet, which, using one feature per node, performed a series of what would now be termed as message passing steps. In each step the feature associated with a node was updated through summing features from all of its neighbors, weighting them differently depending on their distance (Kireev, 1995, Eq. 1).

More recently, following their rediscovery, there has been a large explosion of interest in graph neural network techniques, leading to a proliferation of different architectures, algorithms, and application use-cases. For a review covering recent approaches, we recommend those of Battaglia et al. (2018) and Bronstein et al. (2017) to the reader.

These different variants of GNNs have been introduced from an assortment of distinct, initial viewpoints (Hamilton, 2020, Chap. 7). For example, we described in the main text how GNNs have been proposed as a parameterizable and differentiable molecular fingerprint extractor (Duvenaud et al., 2015; Kearnes et al., 2016). Elsewhere, GNNs have also been presented using analogies to (a) the Weisfeiler-Lehman (WL) graph isomorphism test (Jin et al., 2017; Morris et al., 2019; Xu et al., 2019), (b) inference in probabilistic graphical models (Dai et al., 2016), and (c) spectral convolutions on graphs (Bruna et al., 2013; Defferrard et al., 2016; Kipf and Welling, 2017).

[7] An interested reader can also refer to Battaglia et al. (2018, pp. 14-18 & 36-38) and Gilmer et al. (2017, § 2), which provide an overview of several recent variants, framing them as examples of larger-encompassing architectures named "graph networks" and "Message Passing Neural Networks (MPNNs)" respectively.

In addition to inspiring alternative architectural variants, the viewpoints described above can also inspire different types of theoretical analysis. This theoretical analysis can capture the limitations of current approaches as well as provide pointers towards design improvements. For example, studies have related the representational capacity of GNNs to the 1-dimensional WL test (Morris et al., 2019; Sato, 2020; Weisfeiler and Lehman, 1968; Xu et al., 2019), a method to approximately check for graph isomorphism (informally, whether two graphs are identical). This can be used to show that current GNNs will produce identical molecule embeddings for different molecules, if these same molecules cannot be differentiated using the WL test. An example of such a pathological pair of molecules, admittedly contrived, is dicyclopropylmethane and norbornane, shown here on the right (see Appendix A.2). Fortunately, meeting such pathological pairs in practice is often rare.



dicyclopropylmethane *norbornane*

Architectural or algorithmic innovations for GNNs have also been introduced to address more commonly-occurring, practical problems. Examples include work looking at how to scale up GNNs to operate on larger graphs (Hamilton et al., 2017, § A), as well as work enabling the sharing of information in GNNs over greater distances through virtual nodes or edges (Gilmer et al., 2017, § 5.2). These innovations have also often been driven by adapting techniques to the graph domain that have proved fruitful elsewhere, such as pooling (Lee et al., 2019b), attention (Monti et al., 2017; Veličković et al., 2018), hypernetworks (Brockschmidt, 2020), and the pre-training of models (Hu et al., 2020b).

This proliferation of different graph neural networks leaves the practitioner with a problem: which one to use! Although, various studies have shown that graph neural networks improve upon using fixed fingerprints (Duvenaud et al., 2015, Table 1; Yang et al., 2019; Wu et al., 2018), the relative performance of the networks is sometimes hard to judge, with conflicting evidence presented (Errica et al., 2020; Shchur et al., 2018). Hopefully, this will change through the recent introduction of well-maintained software libraries (Fey and Lenssen, 2019; Wang et al., 2019) and benchmark datasets (Hu et al., 2020a; Wu et al., 2018). For now, when performing experiments in this thesis we will use the Gated Graph Sequence Neural Network (GGNN) proposed by Li et al. (2015). However, the overall architectures and methods we present will be agnostic to this choice, allowing the swapping in of any future advancement in methods able to form molecular and atom embeddings.

2.3 Are 2D representations enough?

The embeddings we saw in the last section were all formed from the 2D graph representation. However, we began this chapter by defining a molecule as existing in *three dimensional* space. Although knowledge of only 2D representations is adequate for understanding the models proposed in the rest of this thesis, this section briefly describes 3D representations to make one better aware of the assumptions we are making, and what modeling molecules as 2D graphs really means.

The particular 3D arrangement of atoms in a molecule is referred to as a "conformation". Conformations can either be derived experimentally, for example through X-ray crystallography, or computationally, for example through quantum mechanical or rule-based methods (Gasteiger and Engel, 2003, § 2.9; Hawkins, 2017). Recently, there has also been the development of ML approaches that predict conformations (Mansimov et al., 2019; Simm and Hernández-Lobato, 2020). These computational methods often take in as input the 2D molecular graph.

Intuitively, we might expect a 3D representation to work better for modeling molecules using ML, as it can more naturally contain data that is missing or often harder to encode in a 2D graph representation, such as stereochemical information. For instance, in the general molecular graph framework we presented in Section 2.1.1, we cannot distinguish between different stereoisomers[8]. We can describe different stereoisomers as existing in different "configurations"; molecules cannot be interconverted between such configurations without the breaking and forming of new chemical bonds, and a particular configuration informs a molecule's chemical properties.

Stereoisomerism is an example of where 3D information is key for distinguishing between different entities with different properties. However, 3D information might even be useful more generally to improve the inductive biases of our models. For instance, in a message passing framework, 3D information could be utilized to enable the passing of messages between atoms that are close in 3D space, even if conceptually far apart in a graph representation.

Therefore, there is a long history of building predictive models that work with 3D molecular and macromolecular structures. This includes methods that, similar to the 2D fingerprint approach seen earlier, first generate fixed-dimensional representation vectors through a pre-determined algorithm, before secondly using these as feature vectors in

[8]Stereoisomerism is the relationship between molecules that have the same atomic composition and connectivity but different spatial arrangements (IUPAC, 2014). Examples include cis-trans isomerism, see Box 3.4, or enantiomerism, the relationship between molecules that are mirror images of one another but non-superposable.

a regular feedforward neural network architecture (Schuur et al., 1996). Moreover, there has also been a series of recent developments in more bespoke neural networks that can directly operate on the 3D structure (Anderson et al., 2019; Ingraham et al., 2019; Schütt et al., 2017; Thomas et al., 2018).

However, despite any intuitions one might have, it is unclear whether the current 3D representations are actually better than 2D representations in practice (Awale and Reymond, 2014; Gao et al., 2020; Sheridan and Kearsley, 2002; Venkatraman et al., 2010). There are several reasons why this is not necessarily as surprising as it may seem. Firstly, as we have already discussed, these 3D conformations might be generated from the 2D molecular graph anyway, meaning that a powerful enough ML model *could*, although not necessarily *would*, learn this transformation if it proved helpful. Secondly, there is not a single conformation for a given molecule; molecules exist in various different conformations with different associated energies, and the one most relevant for a particular effect is often not the one with the lowest energy (Perola and Charifson, 2004). As it can be difficult to model the entire distribution of conformations (a so-called 4D representation), using a subset of them can introduce noise into the modeling system.

In this thesis we therefore stick to using the 2D representations discussed previously. Although they may not be able to model all of the subtleties present, we often find that they provide good performance. Moreover, as was the case when we were discussing which particular GNN architecture to choose (see Box 2.2), the models we shall present are indifferent to the ways in which atom-level or molecule-level embeddings are formed. We can therefore easily update these components at a future date depending on performance and compute requirements.

2.4 Summary

In this chapter we gave a background on the many different ways one can represent molecules to a computer. We explained the different levels of molecular representations and how these implicitly determine the modeling assumptions one makes. We also described the graph neural network, an example of an approach introducing learnable parameters to a model in a structured way to allow it to learn which characteristics are important when forming a higher-level representation. In the chapters that follow, we build upon the modeling of single molecules, described in this chapter, to introduce novel models that can reason with multiple molecules and their interactions, showing such processes can again be considered at different levels of abstraction.

Chapter 3

Reaction Prediction

The reliable prediction of chemical reactions is a key challenge in organic chemistry. It is useful for validating synthesis plans and also of consequence in understanding the processes in molecular biology, that underpin life itself. Moreover, as we shall see in the next chapter, it is also important when designing new molecules for particular tasks.

This chapter describes a new, fully differentiable ML model, ELECTRO, for predicting reactions. ELECTRO approaches the task of reaction prediction from the viewpoint of predicting electron movements (commonly depicted using "arrow-pushing" diagrams). These movements lead to the breaking and forming of chemical bonds, and for a particularly important subset of organic reactions (those with linear electron flow), can be described by a single electron path through the molecules. Instead of predicting product molecules directly, learning a model of electron movement has the benefits of automatically incorporating constraints of chemistry (such as balanced atom counts) as well as being easy to interpret. To train ELECTRO, we develop a technique to extract approximate reaction paths from large reaction datasets. As well as performing well on benchmark tasks, we show ELECTRO learns basic chemical knowledge, such as functional group selectivity, without explicit instruction.

In order to describe ELECTRO, this chapter first starts with a background on chemical reactions, where we explain what it means to "predict" a reaction and provide a description of previous approaches for this task. Then, after explaining ELECTRO in more detail, we evaluate it on a dataset of reactions extracted from the patent literature, and we compare its performance with recent ML models. Before concluding the chapter, we discuss our approach. Here, we highlight ELECTRO's current limitations and avenues for future work, as well as placing it in context with the subsequent work that has been done in this area.

Looking backward, this chapter takes advantage of the node and molecular embedding techniques introduced in the previous chapter. We extend the concepts of how to

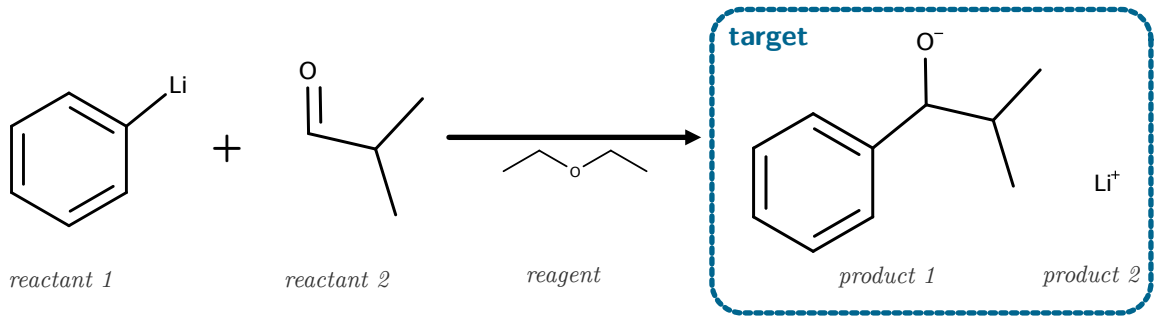


Figure 3.1 *The reaction **product** prediction problem: given the reactant(s) (and reagent(s)) predict the structure of the product(s).*

represent and manipulate single molecules to multiple molecules and their interactions. Looking forward, reaction prediction will form a crucial aspect of the generative models for molecules we will suggest in the subsequent parts of this thesis.

3.1 Background

In this section we give a short background on chemical reactions and computational methods for their prediction. We conclude by describing a fundamental subclass of reactions, those with "linear electron flow" (LEF). We develop a model of LEF reactions, called ELECTRO, which we shall explain in more detail in the next section.

3.1.1 What is a reaction?

What is a reaction? A reaction is "a process that results in the interconversion of chemical species" (IUPAC, 2014). An example of such a process is given in Figure 3.1. However, much like our discussion of molecules in the previous chapter, behind this simple definition lies different levels of abstraction in which we can choose to represent the process. These in turn influence how we can "predict" a reaction. For example, consider Figure 3.1; at the highest level we can regard the reaction shown as a "black box" transform of the reactants to the products, under the context of the reagents[1].

[1] Here we classify reagents as any molecules that are left unchanged at the end of the reaction, but provide context as to how the reaction occurs. Other contextual information, such as the temperature and pressure, we ignore, as it is often absent or intermittently specified in reaction datasets. Where required it can be predicted afterwards using ML techniques, such as those proposed by Gao et al. (2018).

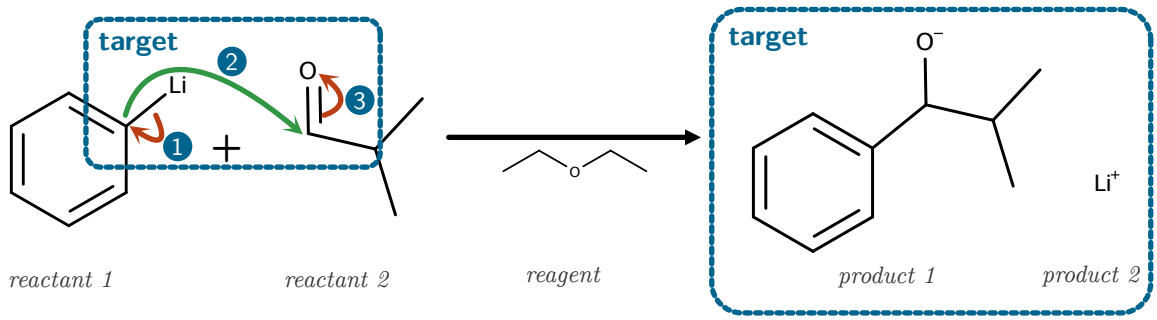


Figure 3.2 *The reaction* ***mechanism*** *prediction problem: given the reactants (and reagents) predict the electron movements involved. These electron movements are depicted using curly arrows; these movements cause the formation (green arrows) and breaking (red arrows) of chemical bonds, and as a consequence, also define the final products formed.*

Such a high-level view, however, neglects the simple chemical constraints present in the process, such as conservation of atom counts and types. Therefore, instead of a black box transform, one could consider a reaction as a graph editing operation.

While using such a representation of a reaction considers *what* has formed, it has "abstracted away" a notion of *how* the reaction occurs, or in other words the "reaction mechanism". The reaction mechanism introduces a notion of time to the reaction process and describes properties of the process that converts the reactants to products. Mechanisms themselves can be treated with different levels of abstraction. On the lowest level we can use quantum mechanics to characterize the system of interest. At a higher level, we can represent the electron movements involved using the commonly used "arrow pushing" notation (also called electron pushing; Kermack and Robinson, 1922; O'Hagan and Lloyd, 2010), as is done in Figure 3.2. Here each "curly arrow" simplifies a complex electron shift, giving the chemist a powerful tool to describe the process. The movement of electrons described by these arrows leads to a series of bond making and bond breaking steps, such that we can also use them to infer the final products.

3.1.2 Reaction prediction

The development of reaction prediction methods often occurs on one or more levels of this hierarchy. At the lowest level, one can use methods performing molecular quantum mechanical calculations to predict the course of a reaction (see e.g. Cramer, 2013; Niu and Hall, 2000). However, often these require a reaction-bespoke setup. Furthermore, these calculations, even when combined with sensible heuristics to limit the amount of computation required, can still be very slow for most systems of interest (Bergeler et al.,

2015; Kim et al., 2018; Nandi et al., 2017; Rappoport et al., 2014; Simm and Reiher, 2017; Socorro et al., 2005; Zimmerman, 2013).

Rule-based approaches

Predominantly rule-based approaches, often able to operate at a higher level of the hierarchy, can make predictions much faster. Early approaches of this ilk include the systems CAMEO (Salatin and Jorgensen, 1980), EROS (Gasteiger et al., 1987), IGOR/RAIN (Ugi et al., 1993, § 4.4; Bauer et al., 1988; Fontain and Reitsam, 1991), and SOPHIA (Satoh and Funatsu, 1995) among others[2]. Although these methods differed in their approach, common among all was the use of expert-encoded rules or heuristics. For instance, CAMEO first used a rule-based categorization procedure to decide on reactive sites and the order of their interactions, before then delegating to mechanism-specific subroutines to carry out actual edits on the reactants.

Research into rule-based systems has continued more recently. For instance, Chen and Baldi (2009) generate a rule-based system where over 1500 reactions are manually encoded. Another method, more recent still, is proposed in Segler and Waller (2017a). Although currently limited to binary reactions, this method is particularly interesting for its ability to suggest new reactions (i.e. new molecules that should react together), through casting the problem as link prediction on a knowledge graph. Using reaction data to discover new reactions is a crucial element of the models that we shall design in the next chapter.

A key feature of many rule-based systems is the use of "reaction templates"; these templates describes how a particular subgraph, using wildcard nodes to describe its wider connectivity, transforms in a reaction. Given a vocabulary of templates and a set of new reactants, the templates can be sequentially trialed, executing any of the specified transforms when the subgraph described in the template is present in the reactants.

Limitations of rule-based systems The early rule-based programs were often limited in scope for computational reasons. For instance, CAMEO was implemented on a Texas Instruments 990/10 computer, with disk storage of just 6M bytes (Salatin and Jorgensen, 1980, Fig. 2). For comparison, the reaction dataset that we use in the later evaluation section of this chapter is roughly 35M bytes in size *when compressed*, and, in terms of

[2]The expert systems for reaction prediction share many similarities with expert systems developed for retrosynthesis, such as LHASA (Corey, 1971; Corey and Wipke, 1969; Corey et al., 1972). Retrosynthesis can be considered the opposite of reaction prediction, in which you are interested in predicting the reactants needed given a product. See e.g.Warr (2014, § 6); Cook et al. (2012); Todd (2005) for a review.

the number of reactions present, it is by no means large compared to some of the private reaction datasets in existence today (Engel and Gasteiger, 2018, § 6.6).

Modern rule-based systems are now often limited for human-driven reasons. Manually defining template libraries is a laborious and hard to scale process, particularly as exceptions to when these rules apply often also have to be recorded (Szymkuć et al., 2016, § 3.2.3). Encoding all possible reactions manually is ultimately a quixotic endeavor. Although automatically extracting templates can help to some extent (Law et al., 2009; Röse and Gasteiger, 1990), when coming to actually use them in practice, it is still slow to enumerate and trial all of those collected. Furthermore, often one can have multiple possible templates that might match, or one template that might match in more than one location[3]. Manually defined priority rules to address this, for instance those used by Chen and Baldi (2009, § 2.3), again require expert input and so are unable to scale.

ML-based approaches

The shortcomings inherent to purely rule-based approaches, have motivated attempts to use ML (machine learning) to predict reaction outcomes. In general we can divide ML reaction prediction methods into two categories: (a) *product prediction*, where the goal is solely to predict the reaction products, given a set of reactants and reagents (see Figure 3.1); and *mechanism prediction*, where the goal is to determine how the reactants react, i.e. the movement of electrons (see Figure 3.2).

Product prediction The drawbacks of only using templates motivated a Templates+ML, hybrid approach (Coley et al., 2017a; Segler et al., 2018b; Segler and Waller, 2017b; Wei et al., 2016; Zhang and Aires-de Sousa, 2005). Here machine learning models can be used for both (a) the preselection of templates (or reaction classes) likely to work, and so avoid an expensive initial enumeration, as well as (b) ranking templates/products by order of preference. These approaches maintain the benefits of using templates, namely their interpretability, but also import a fundamental limit to extrapolation: one will never be able to generalize to reactions that are (even slightly) different to those in the template set.

The WLDN method, proposed by Jin et al. (2017), moves away from using templates and uses ML to model the entire graph edit operation. This method consists of three main steps (illustrated in Jin et al., 2017, Fig. 2): (i) a ML identification of the reaction center (atom pairs that will be involved in the reaction), (ii) enumeration of all chemically possible products by changing the bond configurations in the reaction center, before

[3]These problems are technically referred to as chemoselectivity and regioselectivity – see for instance Clayden et al. (2012, Chap. 23-24).

Table 3.1 *Work on machine learning for reaction prediction, and whether they are (a) end-to-end trainable, and (b) predict the reaction mechanism (i.e. electron movements).*

Prior Work	end-to-end	mechanistic
Templates+ML (e.g. Segler and Waller, 2017b)	–	–
WLDN (Jin et al., 2017)	–	–
Seq2Seq (e.g. Schwaller et al., 2018)	✓	–
Source/Sink (using expert-curated data; Kayala and Baldi, 2011)	–	✓
ELECTRO (this chapter)	✓	✓

finally (iii) a second ML component that ranks all of these possible, enumerated products. For the ML steps, the authors design two graph neural networks (Jin et al., 2017, § 3), the Weisfeiler-Lehman Network and the Weisfeiler-Lehman Difference Network, hence the name WLDN. While leveraging new techniques for learning on graphs, this method is not trained end-to-end due to the enumeration steps required for ensuring chemical validity.

The use of ML methods has also extended to the highest level of reaction abstractions, i.e. treating a reaction as a "black box" transform, through the use of sequence-to-sequence methods (Nam and Kim, 2016; Schwaller et al., 2018; see also Box 3.1)[4]. Opposed to the other ML approaches we have discussed, the sequence-to-sequence approach proposed by Schwaller et al. (2018) (which we shall call Seq2Seq) can be trained end-to-end, or in other words there are not separate ML modules that you need to train and validate independently. However, simple chemical constraints, such as conservation of mass, are no longer "baked-in", but must be learned by the model.

Mechanism prediction ML methods that predict reaction mechanisms include the work of Fooshee et al. (2018); Kayala et al. (2011); Kayala and Baldi (2012, 2011). These models predict the reaction mechanism by identifying interactions between electron sources and electron sinks. While very different to the WLDN method in both the implementation details and even the prediction task being tackled, on a high-level this method follows the same three step process: (i) a ML-based identification of reactive sites, (ii) an enumeration of possible combinations, before finally (iii) a ML-based ranking process. These methods are evaluated on small expert-curated, private datasets, which contain details about reaction conditions, such as the temperature and anion/cation solvation potential (Kayala and Baldi, 2011, § 2). We do not have access to these datasets and so do not compare to these methods here.

[4]Since we first published the work described in this chapter this approach has been further extended by Schwaller et al. (2019)'s Molecular Transformer. The Molecular Transformer now obtains state-of-the-art performance on a range of reaction dataset benchmarks. See Section 3.4.4 for a further discussion.

This work Summarizing the above discussion (see also Table 3.1), there are at least two desirable characteristics for reaction prediction models:

1. *End-to-end*: The space of possible reactions is limited due to complex chemical constraints. How can we pose the problem such that we can differentiate through a model subject to these constraints?

2. *Mechanistic*: Learning the mechanism offers a number of benefits over learning the products directly, including (a) interpretability (if the reaction failed, which part of the electron path is incorrect?); (b) sparsity (often only a small subgraph of the reactant molecules changes during a reaction); and (c) robustness (editing molecules by predicting movements of electrons means that we automatically enforce chemical constraints, such as balanced atom counts).

Given this, we design a model, ELECTRO, that exhibits both of these characteristics. ELECTRO is able to model a subset of mechanisms, those with "linear electron flow" (LEF), which we shall describe next.

3.1.3 Linear electron flow (LEF)

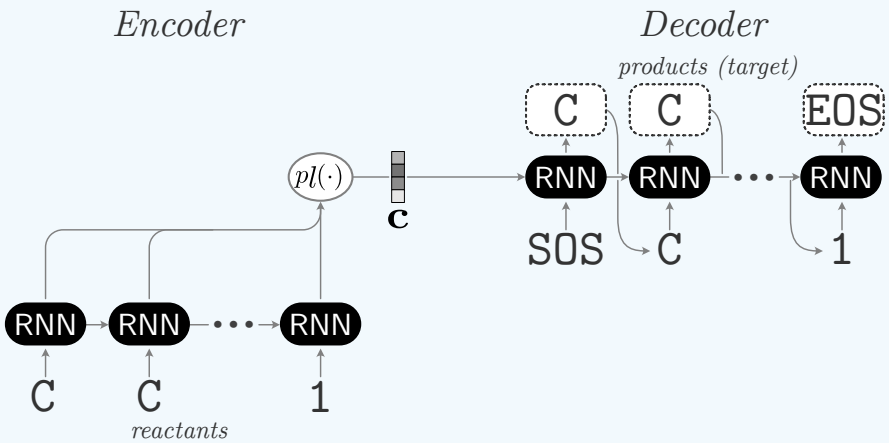
One method to classify reactions is by the topology of their "electron-pushing arrows" (the colored curly arrows in Figure 3.2). Here, the class of reactions exhibiting "linear electron flow" (LEF; Herges, 1994a,b), where the curly arrows line up in sequence, is the most common and central. In this chapter, we shall consider only LEF reactions that are "heterolytic"[5].

In LEF reactions, the electron pairs, described by these arrows, move in a *single* path through the reactant atoms. Further, the electron movements along this path will alternately remove existing bonds and form new bonds. Figure 3.2 shows an example. In this figure, the reaction starts (step 1) by removing the electrons between the Li and C atoms in reactant 1 and moving them to the carbon atom, breaking the bond. Then in step 2, electrons are moved from this C atom to form a bond between this atom and another C atom in reactant 2. Finally (step 3), a pair of electrons are removed from between the C and O atoms, breaking the bond, and moved to the O atom, creating the final products. Notice that by predicting this series of steps (which alternately break and form bonds), we additionally obtain the final product. The description of how we predict these steps, using our model ELECTRO, is the topic of the next section.

[5]Heterolytic reactions are those that involve the movement of pairs of electrons, as opposed to homolytic reactions which involve single electrons (shown in arrow-pushing diagrams as half-barbed arrows).

Box 3.1: Sequence-to-sequence methods

Sequence-to-sequence methods using RNN encoders and decoders were proposed by Cho et al. (2014b); Sutskever et al. (2014). Using variants of the general architecture shown below, these papers demonstrated these methods' utility for language translation tasks (e.g. translating a French phrase into an English one). In these models an encoder RNN is used to embed each token in the input sequence into a vector space. These hidden vector embeddings are then pooled (shown by $pl(\cdot)$) into a context vector $\boldsymbol{c}$, to form an embedding for the entire input phrase. The decoder RNN then uses this context vector to autoregressively construct the output sequence, token by token.



(NB "SOS" and "EOS" denote the "start of sequence" and "end of sequence" tokens respectively)

Later modifications to this model were made to improve its performance. This included removing the pooling operation and introducing an attention mechanism (Bahdanau et al., 2014; Luong et al., 2015). These ideas were further developed by Vaswani et al. (2017) in their Transformer architecture. Here, Vaswani et al. (2017) found that by simplifying previous architectures, in particular by removing recurrent connections and relying only on attention mechanisms, one could obtain even better translation performance, while also reducing the training cost.

Nam and Kim (2016); Schwaller et al. (2018) were the first to explore applying sequence-to-sequence methods to reaction prediction (we compare to the method of Schwaller et al. (2018) in this chapter as the best sequence-to-sequence model existing at the time). This was later improved by Schwaller et al. (2019), in a model that the authors name the Molecular Transformer. They performed a comprehensive evaluation of its performance on a series of tasks, demonstrating state-of-the-art results (Schwaller et al., 2019, p. 1576). Subsequent modifications to the Molecular Transformer have included, among others, introducing latent variables for diversity (Chen et al., 2019), adapting the model for retrosynthesis tasks (Lee et al., 2019a; Schwaller et al., 2020), and using alternative attention mechanisms (Maziarka et al., 2020; Yoo et al., 2020). In these models the SMILES language, introduced in the previous chapter, is used to describe the reactant and product molecules (in reaction SMILES strings the token "≫" is used to separate the reactant from the product). For example, the string "`CC(C)C=O.[Li]C1=CC=CC=C1≫CC(C)C([O-])C1=CC=CC=C1`" would represent the reaction shown in Figure 3.1 (note that often reaction SMILES strings only record the major product(s) – as is done here).

3.2 ELECTRO: a generative model for electron paths

We now describe our probabilistic generative model for the electron movement in LEF reactions. Building upon the notation set up in the previous chapter[6], we shall denote the reactant molecules as $\mathcal{M}_0$ and the product molecules as $\mathcal{M}_{T+1}$. Additionally, sometimes the reaction will occur in the context of reagent molecules, $\mathcal{M}_e$, although these molecules do not themselves change. The reaction happens under a series of electron actions, $\mathcal{A}_{0:T} = (a_0, \ldots, a_T)$, describing the path the electron movements follow. These actions, as discussed in the previous section, will alternately *remove* and *add* bonds (see Figure 3.2), transforming the reactant molecules into the products.

We propose to learn a distribution, $p_\theta(\mathcal{A}_{0:T}|\mathcal{M}_0, \mathcal{M}_e)$, over the electron movements. We choose to factor this distribution autoregressively (a high-level overview of the process is shown in Figure 3.3). At each time step, we want to model, based upon what has gone on before, both (a) the individual action distributions, i.e. the distribution over a_t (describing the electron movements), as well as (b) a distribution over a binary latent variable, c_t (describing if the reaction continues before that step). To be more specific, our distribution is as follows:

$$\begin{aligned} p_\theta(\mathcal{A}_{0:T}|\mathcal{M}_0, \mathcal{M}_e) = & p_\theta^{\text{cont}}(c_0|\mathcal{M}_0)\, p_\theta^{\text{start}}(a_0|\mathcal{M}_0, \mathcal{M}_e) \\ & \times \left[\prod_{t=1}^{T} p_\theta^{\text{cont}}(c_t|\mathcal{M}_t)\, p_\theta(a_t|\mathcal{M}_t, a_{t-1}, t)\right]\left(1 - p_\theta^{\text{cont}}(c_{T+1}|\mathcal{M}_{T+1})\right). \end{aligned} \tag{3.1}$$

We can further break this down into three components: (i) a starting location distribution, $p_\theta^{\text{start}}(a_0|\mathcal{M}_0, \mathcal{M}_e)$; (ii) an electron movement distribution, $p_\theta(a_t|\mathcal{M}_t, a_{t-1}, t)$; and (iii) a reaction continuation distribution, $p_\theta^{\text{cont}}(c_t|\mathcal{M}_t)$. In the subsections that follow we shall define each of these components in turn, before describing how we can train our model and use it for prediction. Algorithm 3.1 provides pseudocode of the entire generative process in the form of a probabilistic program. The exact architectures used in the experiments are described in Appendix B.2.1.

3.2.1 Starting location, $p_\theta^{\text{start}}(a_0|\mathcal{M}_0, \mathcal{M}_e)$

The initial action of the model is to decide which atom, a_0, should start the path. This is conditional on both (a) the initial set of reactants, $\mathcal{M}_0$, and possibly (b) a set of reagents,

[6]Note that in this chapter we are abusing the notation somewhat, allowing $\mathcal{M}$ to refer to multiple molecules at once. This is because in our model we describe the multiple molecules existing at a single step of our process using one graph – the individual molecules themselves will be represented by disconnected subgraphs in this overall graph.

Algorithm 3.1: *The generative steps of* ELECTRO *(given that the model chooses to react, i.e. $c_0 = 1$).*

Input:

- Input molecules, i.e. reactants, $\mathcal{M}_0$, and reagents, $\mathcal{M}_e$;
- GNN to create node embeddings, $\mathsf{GNN}^{\text{node}}(\cdot)$ (i.e. part 1 only of Alg. 2.2);
- GNN to create reagent embedding, $\mathsf{GNN}^{\text{reagents}}(\cdot)$;
- GNN to create continuation logit, $\mathsf{GNN}^{\text{cont}}(\cdot)$;
- Additional NNs to create logits: $f^{\text{start}}(\cdot,\cdot)$, $f^{\text{remove}}(\cdot,\cdot)$, $f^{\text{add}}(\cdot,\cdot)$;
- Max time steps, $T^{\max}$.

Result:

- Electron path, $\mathcal{A}_{0:T}$.

/* We first pick a start location: */

$p_\theta^{\text{start}}(a_0|\mathcal{M}_0,\mathcal{M}_e) \triangleq \mathsf{softmax}\left[f^{\text{start}}\left(\mathsf{GNN}^{\text{node}}(\mathcal{M}_0), \mathsf{GNN}^{\text{reagents}}(\mathcal{M}_e)\right)\right]$

$a_0 \sim p_\theta^{\text{start}}(a_0|\mathcal{M}_0,\mathcal{M}_e)$ ▷ Sample from categorical distribution.

$\mathcal{M}_1 \leftarrow \mathcal{M}_0$ ▷ Molecule does not change until complete pair picked up.

$\mathcal{A}_{0:T} \leftarrow (a_0)$ ▷ Set first location in path.

$c_{t+1} \triangleq 1$ ▷ Do not stop until you have picked up a complete pair of electrons.

▷ Add and remove steps:

for t *in* $[1,\ldots,T^{max}]$ **do**

  if t *is odd* **then**

    $p_\theta^{\text{remove}}(a_t|\mathcal{M}_t,a_{t-1}) \propto \boldsymbol{\beta}^{\text{remove}} \odot \mathsf{softmax}\left[f^{\text{remove}}\left(\mathsf{GNN}^{\text{node}}(\mathcal{M}_t), a_{t-1}\right)\right]$

    $a_t \sim p_\theta^{\text{remove}}(a_t|\mathcal{M}_t,a_{t-1})$ ▷ Sample from categorical distribution.

    $\mathcal{M}_{t+1} \leftarrow \mathsf{remove\_bond}(\mathcal{M}_t, a_t, a_{t-1})$ ▷ Remove bond between a_t and a_{t-1}.

  else

    $p_\theta^{\text{add}}(a_t|\mathcal{M}_t,a_{t-1}) \propto \boldsymbol{\beta}^{\text{add}} \odot \mathsf{softmax}\left[f^{\text{add}}\left(\mathsf{GNN}^{\text{node}}(\mathcal{M}_t), a_{t-1}\right)\right]$

    $a_t \sim p_\theta^{\text{add}}(a_t|\mathcal{M}_t,a_{t-1})$ ▷ Sample from categorical distribution.

    $\mathcal{M}_{t+1} \leftarrow \mathsf{add\_bond}(\mathcal{M}_t, a_t, a_{t-1})$ ▷ Add bond between a_t and a_{t-1}.

  $\mathcal{A}_{0:T} \leftarrow \mathcal{A}_{0:T} \parallel (a_t)$ ▷ Concatenate chosen action onto end of current path.

  /* Assess whether to terminate: */

  $p_\theta^{\text{cont}}(c_{t+1}|\mathcal{M}_{t+1}) \triangleq \sigma\left(\mathsf{GNN}^{\text{cont}}(\mathcal{M}_{t+1})\right)$

  $c_{t+1} \sim p_\theta^{\text{cont}}(c_{t+1}|\mathcal{M}_{t+1})$ ▷ Sample from Bernoulli distribution.

  if $c_{t+1} = 0$ **then**

    break ▷ Reaction stops.

return $\mathcal{A}_{0:T}$

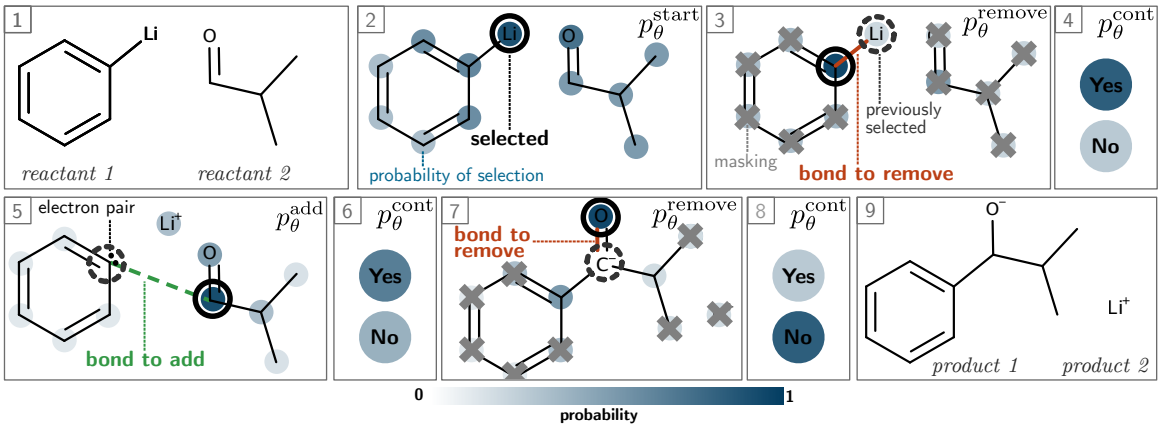


Figure 3.3 *This figure shows the sequence of actions in transforming the reactants in box 1 to the products in box 9. The sequence of actions will result in a sequence of pairs of atoms, between which bonds will alternately be removed and formed, creating a series of intermediate products. At each step the model sees the current intermediate product graph (shown in the boxes) as well as the previous action, if applicable, shown by the gray dashed circle. It uses this to decide on the next action (black circle). We represent the characteristic probabilities the model may have over these next actions as colored circles over each atom. Some actions are disallowed on certain steps, for instance you cannot remove a bond that does not exist; these blocked actions are shown as gray crosses, ✖.*

$\mathcal{M}_e$. We then parameterize $p_\theta^{\text{start}}(a_0|\mathcal{M}_0, \mathcal{M}_e)$ as:

$$p_\theta^{\text{start}}(a_0|\mathcal{M}_0, \mathcal{M}_e) = \text{softmax}\left[f^{\text{start}}\left(\boldsymbol{H}_{\mathcal{M}_0}, \boldsymbol{m}_{\mathcal{M}_e}\right)\right], \tag{3.2}$$

where $f^{\text{start}}(\cdot,\cdot)$ is a feedforward neural network that computes logits for each atom, which are then normalized into probabilities by the softmax function, $\text{softmax}[\boldsymbol{x}] = e^{\boldsymbol{x}}/\sum_i e^{x_i}$. This network takes in the stacked node-level embeddings of the reactants, $\boldsymbol{H}_{\mathcal{M}_0}$, and the graph-level embedding of the reagents, $\boldsymbol{m}_{\mathcal{M}_e}$. These embeddings are computed using the graph neural networks $\text{GNN}^{\text{node}}(\cdot)$ and $\text{GNN}^{\text{reagents}}(\cdot)$ respectively.

3.2.2 Electron movement, $p_\theta\left(a_t|\mathcal{M}_t, a_{t-1}, t\right)$

The next component we shall discuss is $p_\theta(a_t|\mathcal{M}_t, a_{t-1}, t)$, a distribution over where the electrons move next. These movements can be categorized into two classes, depending on whether we are on a remove bond or add bond step.

We learn a separate distribution for these two different classes of action. Both of these two distributions are conditioned on (a) the intermediate molecules formed by the action path up to that point (summarizing the history of previous actions taken), and (b) the immediate previous action a_{t-1} (indicating the immediate previous point on the path).

We can determine which one of our two distributions to use from the parity of t. All LEF reactions start with a remove bond step (as first electrons have to be picked up), and then alternate between removing and adding bonds (§ 3.1.3). Therefore, if t is odd we are on a remove bond step, and vice versa for add bond steps.

Putting this all together we can formally define the electron movement distribution, $p_\theta(a_t|\mathcal{M}_t, a_{t-1}, t)$, as:

$$p_\theta(a_t|\mathcal{M}_t, a_{t-1}, t) = \begin{cases} p_\theta^{\text{remove}}(a_t|\mathcal{M}_t, a_{t-1}) & \text{if } t \text{ is odd,} \\ p_\theta^{\text{add}}(a_t|\mathcal{M}_t, a_{t-1}) & \text{otherwise.} \end{cases} \tag{3.3}$$

$$p_\theta^{\text{remove}}(a_t|\mathcal{M}_t, a_{t-1}) \propto \boldsymbol{\beta}^{\text{remove}} \odot \mathsf{softmax}\left[f^{\text{remove}}\left(\boldsymbol{H}_{\mathcal{M}_t}, a_{t-1}\right)\right], \tag{3.4}$$

$$p_\theta^{\text{add}}(a_t|\mathcal{M}_t, a_{t-1}) \propto \boldsymbol{\beta}^{\text{add}} \odot \mathsf{softmax}\left[f^{\text{add}}\left(\boldsymbol{H}_{\mathcal{M}_t}, a_{t-1}\right)\right], \tag{3.5}$$

where $f^{\text{remove}}(\cdot,\cdot)$ and $f^{\text{add}}(\cdot,\cdot)$ are again feedforward neural networks (we defer a detailed description of the architectures used to Appendix B.2.1). These again make use of stacked node-level embeddings, $\boldsymbol{H}_{\mathcal{M}_t}$, computed by the same graph neural network used for the starting action, $\mathsf{GNN}^{\text{node}}(\cdot)$. The vectors $\boldsymbol{\beta}^{\text{remove}}$ and $\boldsymbol{\beta}^{\text{add}}$ are masks, that ensure zero-probability is assigned to certain atoms based on chemical constraints. Specifically, the mask associated with a remove step, $\boldsymbol{\beta}^{\text{remove}}$, assigns zero probability to an atom a_t if there is not a bond between it and the previous atom a_{t-1} (you cannot remove a bond if it does not exist)[7]. The mask associated with an add step, $\boldsymbol{\beta}^{\text{add}}$, simply prevents the previous action from being repeated; this prevents the model from stalling in the same state for multiple time steps.

3.2.3 Reaction continuation/termination, $p_\theta^{\text{cont}}\left(c_t|\mathcal{M}_t\right)$

Finally, we come to describing how we model when the reaction is to terminate. As mentioned earlier, we introduce a binary latent variable, $c_t \in \{0, 1\}$, that describes whether the reaction continues ($c_t = 1$) or terminates ($c_t = 0$) before step t. The distribution over this variable, $p_\theta^{\text{cont}}(c_t|\mathcal{M}_t)$, is parameterized as:

$$p_\theta^{\text{cont}}(c_t|\mathcal{M}_t) = \sigma\left(\mathsf{GNN}^{\text{cont}}(\mathcal{M}_t)\right), \tag{3.6}$$

[7] A small caveat to this is when a reaction begins with a lone-pair of electrons associated with a single atom; in these cases we say that the reaction starts by removing a "self-bond". Therefore, on the first step, $\boldsymbol{\beta}^{\text{remove}}$ allows the selection of $a_1 = a_0$.

where $\mathsf{GNN}^{\text{cont}}(\cdot)$ is a graph neural network, which learns a mapping from intermediate molecules, $\mathcal{M}_t$, to a single logit (i.e. the molecule-level embedding from this particular GNN is one dimensional). Here $\sigma(\cdot)$ represents the sigmoid function, i.e. $\sigma(x) = 1/(1+e^{-x})$.

Two subtle points in this procedure's implementation need some further clarification. Firstly, for computational reasons we set an upper limit to the number of time steps possible. This is denoted $T^{\max}$ and, in effect, means we are setting $c_{T^{\max}+1} = 0$. Secondly, we cannot stop after the first step, i.e. $c_1 = 1$. This is due to physical constraints; a reaction cannot stop (if it occurs at all) until it has picked up an entire electron pair.

3.2.4 Training and inference

We learn the parameters of our model, θ, by maximizing the log likelihood of a full path, $\log p_\theta(\mathcal{A}_{0:T}|\mathcal{M}_0, \mathcal{M}_e)$. All components of our model are trained at once, including the GNNs computing node embeddings. This training process is done using a form of teacher forcing (Goodfellow et al., 2016, § 10.2.1, Williams and Zipser, 1989, § 2.2); the complete electron path and the associated intermediate products needed for this process are extracted from the training data. Other training techniques could also be used, such as scheduled sampling (Bengio et al., 2015).

At prediction time, given initial reactants $\mathcal{M}_0$ and reagents $\mathcal{M}_e$, we could sample chemically-valid paths from our model by sampling from the appropriate conditional distributions until we sample a continue value equal to zero. However, for making predictions it is often better to find a ranked list of the top-K most likely paths. To do this we use a version of beam search, with width K and maximum path length $T^{\max}$, and in which we record all paths that have terminated.

3.3 Evaluation

In this section we evaluate ELECTRO. To train and test our model we use the USPTO dataset, a collection of chemical reactions extracted from the US patent database (Lowe, 2012). This dataset is not immediately in a form suitable for training ELECTRO, so we first describe how we can preprocess it by extracting approximate reaction mechanisms[8]. We

[8]While the procedure we describe can reconstruct the correct arrow pushing diagram for certain reactions (e.g. S_N2 reactions), given the format of the data, it might simplify the process for other more complicated reactions (e.g. see Figure 3.5), such that we might best describe the mechanism extracted as an "approximate" one. Nevertheless, capturing the overall electron shift, through using these approximate mechanisms, allows us to train our model on large, noisy reaction datasets and compare to state-of-the-art (at the time) ML models for product prediction (§ 3.3.3).

then go on to evaluate ELECTRO on the tasks of (i) *mechanism prediction* and (ii) *product prediction*.

3.3.1 Reaction mechanism identification from USPTO

To evaluate our model, we use a collection of chemical reactions extracted from the US patent database (Lowe, 2012), referred to as the USPTO dataset. We take as our starting point the 479,035 reactions used by Jin et al. (2017) (along with their training, validation, and testing splits). This data consists of a list of reactions. Each reaction is a reaction SMILES string and a list of bond changes. A reaction SMILES string is an extension of the SMILES format we discussed in Section 2.1.3, where an additional token, "≫", is used to split up the molecules present at the beginning of the reaction from those present at the end. These strings have also been atom-mapped, such that a unique integer equates each atom in the reactants with the equivalent atom in the products. The list of bond changes tells us which pairs of atoms have different bonds in the reactants versus the products (note that this can be directly determined from the reaction SMILES string). Below, we describe two data processing techniques that allow us to identify reagents, reactions with LEF topology, and extract an underlying electron path (i.e. an approximate reaction mechanism). Each of these steps can be easily implemented with the open-source chemoinformatics software RDKit (RDKit Team, 2019).

Reactant and reagent separation The reaction SMILES strings can be split into three parts: reactants, reagents, and products. The reactant molecules are those which are consumed during the course of the chemical reaction to form the product, while the reagents are any additional molecules which provide context under which the reaction occurs (for example, catalysts), but do not explicitly take part in the reaction itself; an example of a reagent is shown in Figure 3.1.

Unfortunately, the USPTO dataset as extracted does not differentiate between reagents and reactants. We elect to preprocess the entire USPTO dataset by separating out the reagents from the reactants using the process outlined in Schwaller et al. (2018, § 3.1), where we classify as a reagent any molecule for which either (a) none of its constituent atoms appear in the product, or (b) the molecule appears in the product SMILES string completely unchanged from the pre-reaction SMILES string. This allows us to properly model molecules which are included in the dataset but do not materially contribute to the reaction.

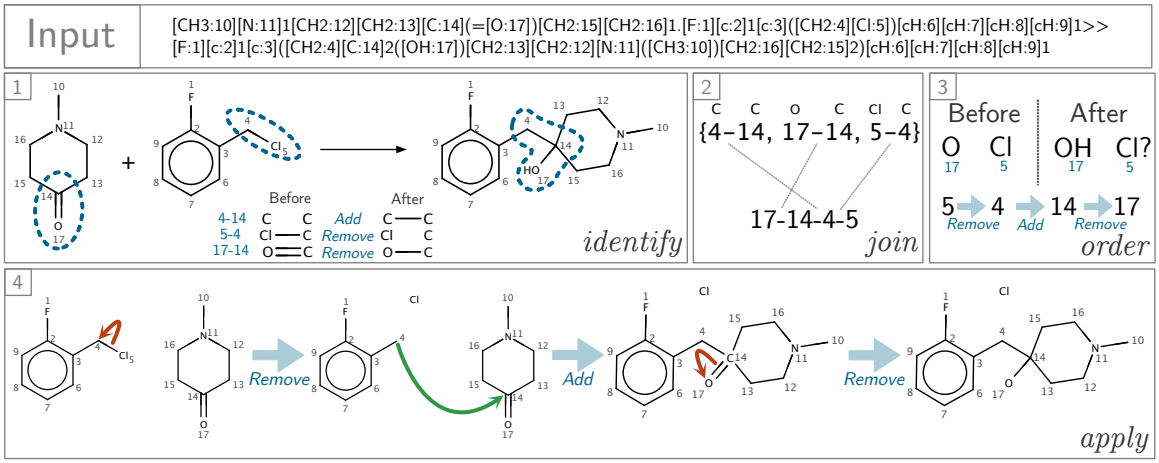


Figure 3.4 *Example of how we turn a SMILES reaction string into an ordered electron path (i.e. an approximate reaction mechanism), for which we can train* ELECTRO *on. This consists of a series of steps: (1) Identify bonds that change by comparing bond triples (source node, end node, bond type) between the reactants and products. (2) Join up the bond changes so that one of the atoms in consecutive bond changes overlap (for reactions which do not have linear electron flow topology, such as multi-step reactions, this will not be possible and so we discard these reactions). (3) Order the path (i.e. assign a direction). A gain of charge (or analogously the gain of hydrogen as* H^+ *ions without changing charge, such as in the example shown) indicates that the electrons have arrived at this atom; and vice versa for the start of the path. When details about both ends of the path are missing from the SMILES string we fall back to using an element's "electronegativity" to estimate the direction of our path; more electronegative atoms attract electrons towards them and so are set as the final atom on our path. (4) The extracted electron path deterministically determines a series of intermediate molecules which can be used for training* ELECTRO*. Paths that do not consist of alternate add and removal steps and/or do not result in the final recorded product do not exhibit LEF topology and so can be discarded. An interesting observation is that our approximate reaction mechanism extraction scheme implicitly fills in missing reagents (caused by noisy training data) – in this example, which is a Grignard- or Barbier-type reaction, the test example is missing a metal reagent (e.g. Mg or Zn). Nevertheless, our model is robust enough to predict the intended product correctly (Effland et al., 1981).*

Identifying reactions with linear electron flow topology To train our model, we need to (i) identify reactions in the USPTO dataset with LEF topology, and (ii) have access to an electron path (i.e. mechanism) for each such reaction. Figure 3.4 shows the steps necessary to identify and extract the electron paths from reactions exhibiting LEF topology. An additional description of these steps is provided in Appendix B.1. Applying these steps, we discover that 73% of the USPTO dataset consists of LEF reactions (349,898 total reactions, of which 29,360 form the held-out test set).

3.3.2 Reaction mechanism prediction

We now start by evaluating ELECTRO on the task of mechanism prediction. Mechanism prediction involves ensuring that we correctly obtain the exact sequence of electron steps. This can be measured, in terms of accuracy, by checking whether the sequence of integers extracted from the raw data (as described in the previous subsection) is an exact match with the sequence of integers output by ELECTRO. Table 3.2 shows the top-1, top-2, top-3, and top-5 accuracies computed in this manner.

In this table we compare against a variant of our model called ELECTRO-LITE. ELECTRO-LITE ignores the reagent information, $\mathcal{M}_e$, when making predictions. Although reagent information is generally required to faithfully predict a reaction, it can often be inferred from the reactants alone, and this ablation study allows us to gauge its importance.

We also perform a more qualitative analysis on the mechanisms predicted. Complex molecules often feature several, competing functional groups that could react. Chemists can use heuristics and trends, derived from experimental observation, to predict the "selectivity", i.e. which of these functional groups will predominantly react in the presence of the others. By plotting the predictions of ELECTRO on a series of textbook reactions, shown in Figure 3.5 (and in Appendix B.3), we can study whether ELECTRO has learned the same trends from data. Overall, we found that the model's predictions for most examples were correct; in the small quantity of incorrect cases, interpreting the model's output reveals that it still made chemically plausible predictions.

3.3.3 Reaction product prediction

Reaction mechanism prediction is useful to ensure we form the correct products in the "correct" way. However, it underestimates the model's actual predictive accuracy: although a single atom mapping is provided as part of the USPTO dataset, in general atom mappings are not unique (for an example see Box 3.2).

Table 3.2 *Results when using* ELECTRO *for* ***mechanism prediction****. Here a prediction is correct if the atom-mapped action sequences predicted by our model exactly match those extracted from the USPTO dataset.* ELECTRO-LITE *is a variant of our model that ignores reagent information, acting as an ablation study to gauge the importance of this information.*

	Accuracies (%, ↑)			
Model Name	Top-1	Top-2	Top-3	Top-5
ELECTRO-LITE	70.3	82.8	87.7	92.2
ELECTRO	77.8	89.2	92.4	94.7

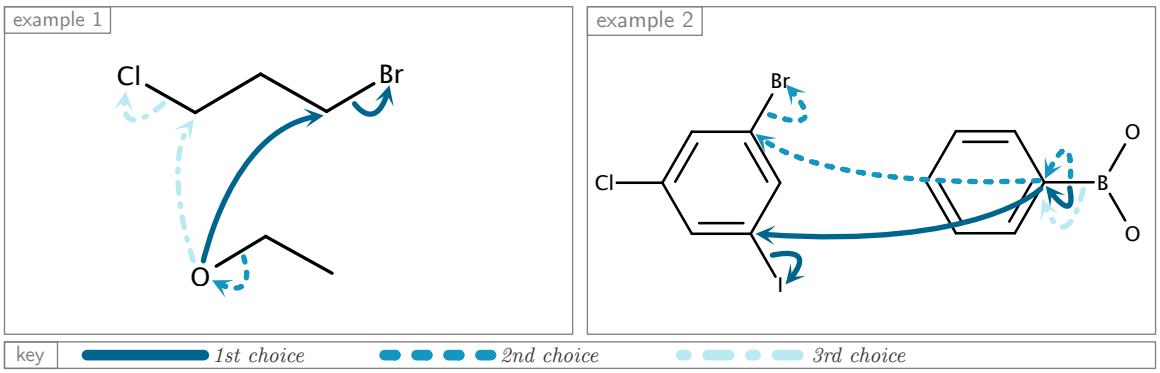


Figure 3.5 *Example 1, on the left: nucleophilic substitution S_N2-reaction. Example 2, on the right: Suzuki-coupling. (Note that in the "real" mechanism of the Suzuki coupling, the reaction would proceed via oxidative insertion, transmetalation and reductive elimination at a palladium catalyst. As these details are not contained in training data, we treat palladium implicitly as a reagent). In both cases, our model has correctly picked up the trend that halides lower in the period table usually react preferably ($I > Br > Cl$).*

Recent approaches to *product prediction* (Jin et al., 2017; Schwaller et al., 2018) have evaluated whether the major product reported in the test dataset matches predicted candidate products generated by their system, independent of mechanism. In our case, the top-5 accuracy for a particular reaction may include multiple different electron paths that ultimately yield the same product molecule.

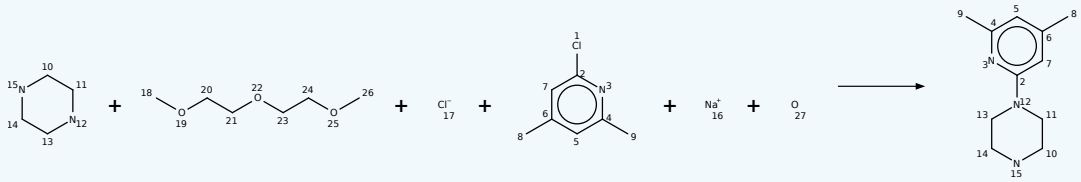
To evaluate if our model predicts the same major product as the one in the test data, we need to solve a graph isomorphism problem. To approximate this, we (i) take the predicted electron path, (ii) apply the induced edits to the reactants to produce a product graph (balancing charge to satisfy valence constraints), (iii) remove atom mappings, and (iv) convert the product graph to a canonical SMILES string representation in Kekulé form (aromatic bonds are explicitly represented as double bonds, see § 2.1.1). We can then evaluate whether a predicted electron path matches the ground truth by a string comparison. This procedure is inspired by the evaluation of Schwaller et al. (2018). To obtain a ranked

Table 3.3 *Results for **product prediction**, following the product matching procedure in Section 3.3.3. For the baselines we compare against models trained (a) on the full USPTO training set (marked FTS) and only tested on our subset of LEF reactions, and (b) those that are also trained on the same LEF subset of USPTO as our model. We make use of the code and pre-trained models provided by Jin et al. (2017). For the Seq2Seq approach, as neither code nor more fine grained results are available, we train up the required models from scratch using the OpenNMT library (Klein et al., 2017).*

	Accuracies (%, ↑)			
Model Name	Top-1	Top-2	Top-3	Top-5
WLDN FTS (Jin et al., 2017)	84.0	89.2	91.1	92.3
WLDN (Jin et al., 2017)	83.1	89.3	91.5	92.7
Seq2Seq FTS (Schwaller et al., 2018)	81.7	86.8	88.4	89.8
Seq2Seq (Schwaller et al., 2018)	82.6	87.3	88.8	90.1
ELECTRO-LITE	78.2	87.7	91.5	94.4
ELECTRO	**87.0**	**92.6**	**94.5**	**95.9**

Box 3.2: Limitations in mechanism-based evaluation

A mechanistic-based evaluation (as described in §3.3.2) can underestimate a model's predictive accuracy, as different electron paths can form the same products. This might occur for example due to symmetry. Consider the reaction, taken from USPTO, drawn below:



Although only one electron path is given in the USPTO dataset, due to symmetry, we could change the atom-mapping without changing the course of the reaction in any meaningful way. This is shown below:

(a) **(b)**

Therefore, an assessment based purely on electron path accuracy can sometimes be misleading, and, as such, can be complimented by product-based assessments (§ 3.3.3).

list of products for our model, we compute this canonicalized product SMILES string for each of the predictions found by beam search over electron paths, removing duplicates along the way. These product-level accuracies are reported in Table 3.3

We compare with the graph-based method Jin et al. (2017) (shown to be better than previous template-based methods); we use their evaluation code and pre-trained model, re-evaluated on our extracted test set (more details on all the baselines are provided in Appendix B.2.2). We also use their code and re-train a model on our extracted training set, to ensure that any differences between our method and theirs is not due to a specialized training task. We also compare against the Seq2Seq model proposed by (Schwaller et al., 2018); however, as no code is provided by Schwaller et al. (2018), we run our own implementation of this method based on the OpenNMT library (Klein et al., 2017). Overall, ELECTRO outperforms the other approaches on this task, with 87% top-1 accuracy and 95.9% top-5 accuracy. Omitting the reagents in the ELECTRO-LITE model described earlier, degrades top-1 accuracy slightly, but maintains a high top-3 and top-5 accuracy. This suggests that reagent information is necessary to provide context in disambiguating plausible reaction paths.

3.4 Discussion

In this section we discuss the approach, its limitations and possible extensions, and put it into context with the subsequent work done in this area since its initial publication.

3.4.1 Beyond LEF reactions

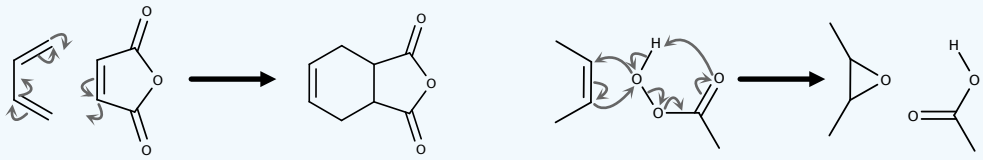
Perhaps the most obvious limitation of ELECTRO is its restriction to LEF reactions. Although, as we discussed earlier, this encompasses the majority of reactions (Herges, 1994b), it would still be beneficial to extend the model to cover other reaction classes (see Box 3.3) and reactions consisting of multiple stages[9]. To do this, two challenges must be overcome: (a) modeling these other reaction classes, and (b) obtaining suitable training data for these other reaction classes. We shall now discuss both of these challenges in turn.

The first challenge of modeling other reaction classes could be overcome in a variety of different ways. For instance, pericyclic reactions (see Box 3.3) could be transformed

[9]Such a reaction might often be termed a stepwise reaction (IUPAC, 2014). We have used the term "stage" here to avoid overloading the term "step", particularly to avoid confusion with the individual electron steps in the autoregressive prediction process of ELECTRO.

Box 3.3: Electron flow topologies

There are many different approaches to classify reactions (Gasteiger, 2003, Chap. III). One way is by the topology of the electron pushing arrows involved (Herges, 1994a,b). In this manner, Herges (1994b, Fig. 1) broke down a large reaction dataset into three main classes: (1) linear, (2) cyclic, and (3) complex (coarctate). An example of a linear topology, the main focus of this chapter, has previously been shown in Figure 3.2. An example of the cyclic and complex topologies is shown below:



(a) *A Diels-Alder reaction, see e.g. Clayden et al. (2012, pp. 879-886); an example of a* ***cyclic*** *topology.*

(b) *An epoxidation reaction, see e.g. Clayden et al. (2012, pp. 429-430); an example of a* ***complex*** *topology.*

(NB we are drawing an arrow for every bond add and remove step to match our presentation elsewhere and to remove ambiguity on where the new bonds form; these diagrams can also be simplified in presentation by merging these arrows such that each goes from the bond that has been removed to the bond that has been added.)

Linear reactions, such as those modeled by ELECTRO, have arrows lining up. Cyclic reactions (also called pericyclic if the electron movements are concerted, i.e. occur in unison) have a cyclic electron path with no defined start or end. Both of these classes have one bond added and/or removed at each atom during the reaction. On the other hand, complex reactions have at least one atom at which two bonds are added or removed (Herges, 2015).

These reaction classes have different rates of occurrence in common reaction databases; the LEF class is often the most common. For instance, Herges (1994a, Figure 2), when considering a large database of approximately 80,000 reactions, found 80% of the reactions were of this class, with the cyclic class being the second most common.

into a LEF like flow by arbitrarily breaking the cycle to form a path, and then randomly assigning a direction to this path.

We also need to deal with multi-stage reactions consisting of sequential elementary LEF reactions that do not form a single path. Practically, this is a simpler problem and could be solved by allowing the products to be fed back through ELECTRO, possibly several times.

The second challenge, that is of obtaining suitable training data, is possibly more challenging. The technique we introduced for obtaining approximate mechanisms for LEF reactions (§ 3.3.1) does not currently work for more complex reactions. One conceivable approach is to use finer grained datasets that include mechanistic intermediates or the mechanisms themselves (e.g. the private datasets used by Kayala and Baldi, 2011). Alternatively, these mechanistic intermediates could be created using a hybrid approach, based on both ML and quantum mechanical calculations (Pattanaik et al., 2020b; Sadowski et al., 2016).

3.4.2 Repurposing for retrosynthesis

Single step retrosynthesis can be seen as the opposite to reaction prediction. In this task one is given the products and is required to predict the reactants needed to create them (Coley et al., 2017b; Dai et al., 2019; Segler and Waller, 2017b; Shi et al., 2020; Somnath et al., 2020). One main use of this is as a component in computer-aided synthesis planning (CASP) tools (see e.g. Chen et al., 2020; Segler et al., 2018b; Szymkuć et al., 2016; Todd, 2005), which plan (often over several reactions) how to break a complex molecule down into simple and available "building block" molecules.

In principle, one could simply adapt ELECTRO to the single step retrosynthesis task by training it on reversed electron paths. In practice, however, further challenges may arise. In particular, in retrosynthesis tasks you are often only given the "major product", whereas the reactants will also contain atoms that may have ended up in the "minor products". This means that purely graph-editing methods, such as ELECTRO, are ill posed to tackle this problem.

One possible approach to this problem is to augment such graph-editing methods with an auxiliary model that can predict the minor products or at least the important subgraphs of such molecules, and so introduce new atoms into the backwards task. This is in fact one of the advantages of sequence-to-sequence–based models, which are able to do this implicitly (Lee et al., 2019a).

3.4.3 The 3D question (again)

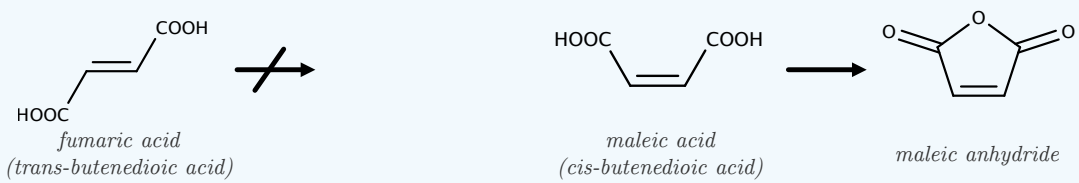
We discussed in the previous chapter (§ 2.3) the limitations inherent in using 2D representations for molecules. While we do not wish to repeat that entire discussion here, it is worth briefly pointing out how these ideas relate to reactions.

Critical to this is the idea of stereoselectivity, the favorable formation of one stereoisomer over another in a reaction (IUPAC, 2014; see also Box 3.4). Models, such as ELECTRO,

Box 3.4: Stereoselectivity

When distinguishing between molecules we can do so at the level of the "constitution" or the "configuration" (or even the level of the "conformation" – see § 2.3). The constitution describes which atoms exist in a molecule and how they are connected. However, it ignores any distinction occurring from the atoms' spatial arrangement. Introducing stereochemical information allows us to describe the configuration of a molecule. While having the same connectivity, molecules with different configurations cannot be interconverted without breaking and creating new chemical bonds (note that a single *configuration* can still exist in multiple *conformations*, for example through rotation around a single bond).

An example of molecules with different configurations are those that are cis-trans isomers (also called E/Z isomers). These isomers occur when atoms take different positions relative to a reference plane (IUPAC, 2014). Cis-trans isomers can have very different chemical properties. For example, due to the lack of rotation about a carbon-carbon double bond (in turn because of the molecular orbitals involved), maleic and fumaric acids have very different melting points, boiling points, and even solubilities. They also act very differently under heat (Clayden et al., 2012, p. 105; Musa, 2016, Chap. 2); maleic acid reacts, losing water in a dehydration reaction, but fumaric acid does not, as shown below:



making use of GNNs that cannot distinguish between molecules with different configurations will not be able to currently model these reactions correctly. This could be fixed by using newer GNNs that work with this information (Pattanaik et al., 2020a) as well as datasets in which this information has not already been stripped out (Schwaller et al., 2019, p. 1573).

3.4.4 Subsequent reaction prediction methods

Given the exciting current pace of ML research, there has already been further development of approaches for reaction prediction since we initially published[10] the work described in this chapter. For example, the WLDN and Seq2Seq models (used in the experiments of § 3.3.3), state-of-the-art at the time, have since been further developed in

[10] As a preprint in 2018.

Coley et al. (2019) and Schwaller et al. (2019). In fact, we will use Schwaller et al. (2019)'s Molecular Transformer (see Box 3.1) as a reaction predictor component in the models we present in the subsequent parts of this thesis, due to its improved accuracy (see Schwaller et al., 2019, Table 4) and scope for predicting non-LEF reactions.

The notion, espoused by models such as ELECTRO, of representing a reaction as a *sequence* of graph edits has also been adopted and developed further. Examples include MEGAN (Sacha et al., 2020) and GTPN (Do et al., 2019). However, these methods have moved away from the philosophy of trying to model electron flows. Another new approach, proposed in Bi et al. (2020), predicts electron flows, but does so in parallel.

Generally, the evaluation of these models has continued to be done in a similar manner to the approach used in Section 3.3.3 (which in turn is based on the evaluation done in Jin et al., 2017); that is to say, models are compared using a top-K accuracy metric on versions of the USPTO dataset. In the future it would be interesting to compare other aspects of reaction predictor models, such as their ability to generalize to new domains, using for instance time-based or reaction fingerprint–based splits of the data.

3.5 Summary

In this chapter we described ELECTRO. ELECTRO approached the problem of reaction prediction not as a generic graph edit operation but one that is inferred through the modeling of movements of electrons. We applied ELECTRO to a prediction task derived from the USPTO dataset, and we found it compared favorably to the baselines of the time. Moreover, it was able to learn basic chemical trends without implicit instruction. Reaction prediction is important for better understanding chemical processes or validating synthesis plans. Furthermore, as we show in the next chapter, it can play a crucial role in the construction of new molecules.

Chapter 4

Searching for Synthesizable Molecules

When designing a new molecule with particular properties, it is not only important what to make but also, crucially, *how to make it*. This is the case irrelevant of what the molecule is ultimately for: new drugs for the world's healthcare, new agrochemicals for the world's food, and even new efficient materials for the world's energy demands *all* have to be made. The instructions for how to make a molecule take the form of a synthesis plan: in effect a recipe for a particular molecule, where simple building block molecules get combined together to form more complicated molecules through chemical reactions, such as those we described in the last chapter.

Recently, there has been tremendous progress in designing new ML (machine learning) methods for generating and optimizing molecules for particular tasks. However, these improvements in molecule discovery have come at a cost: these methods often do not describe how to *synthesize* such molecules, a prerequisite for experimental testing. Therefore, in this chapter, we design a model called MOLECULE CHEF that, aiming to better reflect the real world process, builds molecules up through (virtual) chemical reactions. More specifically, it does so through a two-step process, first selecting a "bag" (i.e. multiset) of reactants, before secondly feeding these through a reaction predictor to generate a final product.

We argue that constructing molecules through chemical reactions provides a sensible inductive bias to our model. It allows us to combine the strengths of modern ML approaches, particularly their capacity for organizing and for effectively searching over chemical space, with those of older techniques, such as virtual synthesis, where reactions are explicitly considered. It also provides interpretability, allowing chemists to interrogate not only the molecular outputs of our model but also the feasibility of the synthetic routes. To demonstrate the advantages of our approach, we evaluate MOLECULE CHEF on a series of tasks and show that MOLECULE CHEF can (i) generate a wide range of diverse and valid

structures not seen in the training data, (ii) optimize molecules for particular properties, and (iii) propose new reactants for a given product molecule that may be easier to manage.

To describe MOLECULE CHEF in more detail, this chapter is split into four main parts. We first provide a background on traditional computational techniques to discover new molecules, setting the scene for MOLECULE CHEF, which we then describe in more detail in the second section. We evaluate MOLECULE CHEF on the tasks of generation, optimization, and retrosynthesis in the third section, before briefly summarizing the conclusions of this chapter in the fourth and final section. This chapter draws on material presented in both of the previous chapters, with graph neural networks and reaction predictors forming a key part of MOLECULE CHEF's architecture. Looking forward, the subsequent chapter extends the ideas of MOLECULE CHEF to multiple steps of reactions.

4.1 Background

Deep generative models of molecules draw ideas from both the field of traditional molecule design (see e.g. Hartenfeller and Schneider, 2011) and also the field of generative modeling of structured data (see e.g. Goodfellow et al., 2016, Chap. 20). Reflecting this parentage, this background starts with an introduction to molecule design, discussing traditional approaches and how the recent ML methods relate to these. This background then goes on to discuss the development of ML approaches. This discussion seeks to set this development in the context of two overarching, related questions: (a) what makes a generated molecule "viable", and (b) how can ML methods pose the construction of molecules to ensure such "viability"?

In progressing through this section, we shall make use of boxes to tie the discussion back into older chemoinformatics approaches and ML generative modeling. Finally, this section concludes with our attempt to answer the question of how to generate viable molecules. We do this by introducing, at a high-level, our model MOLECULE CHEF, seeding the way for a more detailed description and evaluation of it in the sections that follow.

4.1.1 In search of new molecules

The search for new molecules usually proceeds in what are commonly referred to as "design-make-test-analyze cycles". Here, given a particular target, new molecules are designed[1], made and then tested in the lab, the results of which are then analyzed to

[1] Although molecules are often referred to as being "designed", the meaning we ascribe to this word might be closer to that of "selected" or even "discovered" (Jansen and Schön, 2006). In this text we will use these words interchangeably.

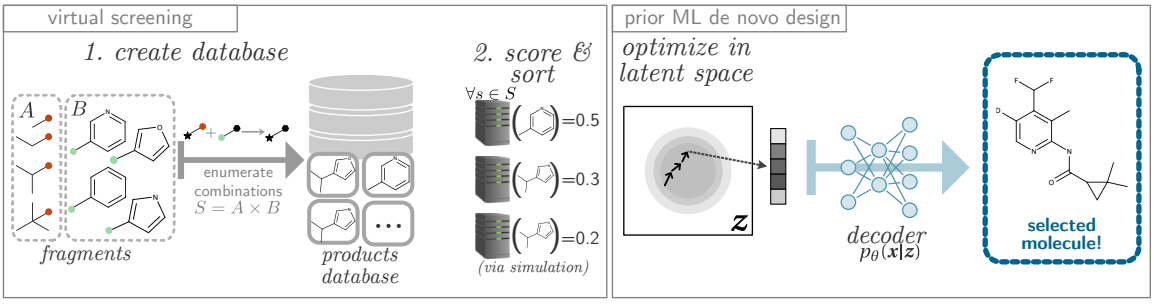


Figure 4.1 *An overview of previous approaches used to find molecules with desirable properties.* Left*: virtual screening (Shoichet, 2004) aims to find novel molecules by the (computationally expensive) enumeration over all possible combinations of fragments.* Right*: more recent ML approaches, e.g. Gómez-Bombarelli et al. (2018), aim to find useful, novel molecules by optimizing in a continuous latent space; however, there are no clues as to whether (and how) these molecules can be synthesized...*

inform the next iteration through the cycle (Holenz, 2016). In the design stage, there are two main computer-assisted approaches: virtual screening (VS) and de novo design (DND; Hartenfeller and Schneider, 2011).

Virtual screening (VS) In virtual screening (Figure 4.1, left), a large database of molecules is first generated. Given a property of interest, this database is then filtered, sometimes using ML techniques (Gómez-Bombarelli et al., 2016; Muratov et al., 2020), to find the best matching molecule. An advantage of this approach is the high level of control the chemist can maintain on the individual molecular constituents of the database. These molecules can be chosen due to their direct availability from chemical vendors or even generated using reaction schemes from easily obtainable building blocks (Chevillard and Kolb, 2015; Humbeck et al., 2018). The disadvantage with VS is its scalability. A database of N molecules takes $\mathcal{O}(N)$ in compute and storage costs, limiting its ultimate size. As a result of this, the largest VS databases are of the order of $|N| \approx 10^8$ in size, which is miniscule in comparison to the space of possible drugs ($\approx 10^{23}$ to 10^{60} in size; Pyzer-Knapp et al., 2015; Reymond et al., 2012; Shoichet, 2004; van Hilten et al., 2019; Walters, 2019; Walters et al., 1998).

De novo design (DND) To achieve higher chemical space coverage, one can instead turn to de novo design. Here instead of designing the individual molecules themselves, one designs a molecule construction/edit operation and couples this with an optimization/search procedure. Early techniques (see Box 4.1) made use of discrete search techniques, such as ant colony optimization or genetic algorithms (GAs; Hiss et al., 2014; Schneider, 2013). However, these techniques often relied on local edit functions, limiting

Box 4.1: Traditional approaches for de novo design

Computer programs for de novo design have existed for over 30 years (Hartenfeller and Schneider, 2011, Table 1). While we will not examine all of these advances here, it is interesting to note that these traditional approaches from the chemoinformatics literature have explored similar themes to recent ML work, particularly in terms of how best to sequentially construct molecules. For example, different approaches have considered constructing molecules atom by atom (Luo et al., 1996; Nishibata and Itai, 1991), by using fragments (DeWitte and Shakhnovich, 1996; Schneider et al., 2000), and also, most relevant to the work in this chapter, by using virtual reaction schemes (Hartenfeller et al., 2012; Vinkers et al., 2003).

These approaches are often tied to older, greedy optimization techniques, limiting their overall performance. For instance, Vinkers et al. (2003)'s SYNOPSIS algorithm gradually expands a pool of synthetically accessible molecules by utilizing a rule-based reaction scheme. More specifically, a simulated annealing method is used to pick a current member of the pool, before a compatible reaction is randomly chosen to change it. We include a comparison of the model we propose in the next chapter (DoG-Gen) to this algorithm in Appendix D.5.3.

both the diversity of the molecules produced and the speed in which one could traverse chemical space.

4.1.2 ML-based de novo design

A paradigm shift in molecule design happened with the introduction of modern ML approaches and the development of generative models of molecules (Figure 4.1, right). In a late 2016 preprint, Gómez-Bombarelli et al. (2018) introduced the CVAE (character-based variational autoencoder) for molecule search. Their idea was to train a variational autoencoder (see Box 4.2) on SMILES strings, learning a mapping between a continuous latent space and the space of molecules. Their model turns a discrete, difficult-to-optimize space into a continuous one, where suddenly the arsenal of continuous-based search techniques can be leveraged for the problem, such as Bayesian optimization (Shahriari et al., 2016) for global search or gradient descent for local search. Elsewhere, Segler et al. (2018a) and Olivecrona et al. (2017) introduced autoregressive models of SMILES strings, pairing these with reinforcement learning (RL) algorithms for optimization; these models have even successfully suggested molecules that have since performed well in lab experiments (Merk et al., 2018a,b; Yang et al., 2020).

Box 4.2: Deep latent variable models

In unsupervised learning, a machine learning model receives as data inputs, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, but no targets (Ghahramani, 2004). Two classic problems in unsupervised learning are (a) learning *representations* of such data, and (b) *generating* new data. We can tackle both problems at the same time using a deep latent variable model.

In a (continuous) latent variable model we associate with each observed input, $\boldsymbol{x}$, a latent "representation" (or "variable"), $\boldsymbol{z} \in \mathbb{R}^d$. This variable should *represent* in some sense the "essence" of the observed data. We model our distribution over observed data as follows:

$$p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}|\boldsymbol{z})\, p(\boldsymbol{z})\, \mathrm{d}\boldsymbol{z}, \tag{4.1}$$

where $p(\boldsymbol{z})$ is a prior over the latent variable, often just a simple Gaussian, and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is the likelihood function, which here we will parameterize using a deep neural network.

Our task then is to ensure that the marginal likelihood, $p_\theta(\boldsymbol{x})$, matches the data distribution, $p_{\text{data}}(\boldsymbol{x})$. This can be done in different ways. For example, the variational autoencoder (VAE; Kingma and Welling, 2013; Rezende et al., 2014) minimizes an upper bound on the KL divergence, $\mathcal{KL}\big(p_{\text{data}}(\boldsymbol{x}), p_\theta(\boldsymbol{x})\big)$. Specifically, it introduces an approximate posterior, $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, and maximizes the following lower bound, $\mathcal{L}_{\text{VAE}}$:

$$\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x})\right] = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\log \int \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\, p_\theta(\boldsymbol{x}|\boldsymbol{z})\, p(\boldsymbol{z})\, \mathrm{d}\boldsymbol{z}\right], \tag{4.2}$$

$$\geq \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log p_\theta(\boldsymbol{x}|\boldsymbol{z}) \frac{p(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\, \mathrm{d}\boldsymbol{z}\right], \quad \text{(by Jensen's inequality)} \tag{4.3}$$

$$= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] - \mathcal{KL}\big(q_\phi(\boldsymbol{z}|\boldsymbol{x}), p(\boldsymbol{z})\big)\right] = \mathcal{L}_{\text{VAE}}, \tag{4.4}$$

with respect to the parameters, θ and ϕ. Here the approximate posterior is often also a Gaussian distribution, the mean and variance of which are given by a deep neural network; using a function to infer the distribution over latent variables in this way is called "amortized inference" (Dayan et al., 1995; Gershman and Goodman, 2014).

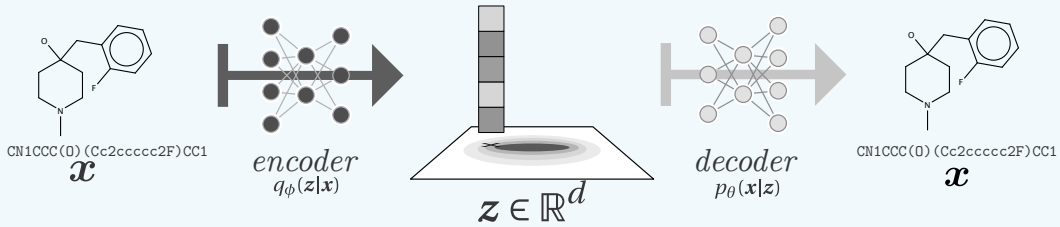
Unfortunately, VAEs can be hard to train, especially when coupled with complex likelihoods, such as those used when $\boldsymbol{x}$ is a molecule. Specifically, they can suffer from a phenomenon called "posterior collapse"; this occurs when the approximate posterior distribution "collapses", mapping all inputs to the prior. While this minimizes the second KL-term of Equation 4.4, the resulting latent representations become meaningless and are ignored by the likelihood model. To avoid this, one can scale the KL term by a factor β (Alemi et al., 2018; Bowman et al., 2016; Higgins et al., 2017), and gradually increase β during training, an approach taken by many previous molecule autoencoders (see e.g. Kusner et al., 2017; Liu et al., 2018). However, to avoid having to tune such a parameter, in this work, we instead maximize a Wasserstein autoencoder (WAE) objective (Tolstikhin

et al., 2018):

$$\mathcal{L}_{\text{WAE}} = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\Big[\mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right]\Big] - \lambda D\Big(\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[q_\phi(\boldsymbol{z}|\boldsymbol{x})\right], p(\boldsymbol{z})\Big), \tag{4.5}$$

where following Tolstikhin et al. (2018), $D(\cdot,\cdot)$ is a maximum mean discrepancy (MMD) divergence measure (Gretton et al., 2012) and $\lambda = 10$. This WAE objective matches the VAE objective in the first term (although more general cost functions can also be used here), encouraging the model to reconstruct the data correctly; however, it differs in the second term, such that one is no longer trying to match the approximate posterior distribution of *each* latent variable to the prior (causing them to overlap), but the marginalized distribution over *all* datapoints. Empirically, we find that training with the WAE objective works well, and does not suffer from the same posterior collapse problem that affects the VAE.

Regardless of which of these losses we ultimately use to train the networks which parameterize the approximate posterior and likelihood distributions, we can depict the full model as follows:



This highlights the relationship to older autoencoder models (see e.g. Hinton and Zemel, 1994, or Goodfellow et al., 2016, Chap. 14). The approximate posterior is used to *encode* observed data into the latent space, and so is often referred to as the "encoder" (or the recognition or inference network). Likewise, the likelihood, $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, is used to *decode* from the latent representation back to the observed variable, and so is called the "decoder". In this manner, the latent space is used to *represent* the data and so we would expect different inputs to get encoded into different areas of latent space. By picking new points in the latent space, through sampling, interpolating, or even optimizing, and then passing these points through the decoder, we can *generate* new data.

In designing deep latent variable models for molecules, the structure of the encoder and decoder is crucial. The encoder, a mapping from a molecule to a real-valued vector, can be built using any of the techniques we discussed in Chapter 2. The decoder, a mapping from a real-valued vector to a molecule, is usually more complicated and can often be autoregressive in nature. For example, Gómez-Bombarelli et al. (2018, p. 274) represented molecules using SMILES strings, using a convolutional neural network for the encoder and a recurrent neural network for the decoder. In this chapter and the next we shall propose new encoders and decoders built around chemical reactions.

Viability of proposed molecules It is worth clarifying here, however, that molecule search is not an unconstrained optimization problem. When designing new molecules using DND approaches, whether that be through an older GA-based or more modern ML-based approach, we need the suggested molecules to be "viable". What makes a suggested molecule "viable"? Unfortunately, this is hard to define. Broadly, it depends on the ideas of (a) validity, (b) stability, and (c) synthesizability. Validity, perhaps the easiest to define, relates to a concept of syntactic correctness. So for instance when generating SMILES strings, validity means ensuring the final string can be parsed, or in other words brackets and rings are closed, the chemical symbols used make sense, and so on. Harder to define is the idea of stability, which instead relates more towards a notion of semantic correctness. This can appear subtle to non-chemists and depends a lot on the task at hand. Particular substructures may be stable in an inert refrigerator but not in the human body! Finally synthesizability, which we shall come back to later, means that the molecule can actually be made.

When designing molecule search techniques, there is an inherent trade-off between chemical space coverage and the viability of the molecules suggested. VS, which we introduced at the beginning of this section, can be seen as being at one end of this trade-off, with limited space coverage but strong control over the viability of the molecules suggested. Inhabiting the other end would be a completely unconstrained de novo design algorithm, for instance one using GAs to connect up any atoms in any manner when constructing the molecular graph.

Initially, it was hoped that ML-based approaches might learn a notion of viability as part of their training process. However, unfortunately, this was often not the case. Particularly when being used for optimization, early models (decoding to SMILES strings) often generated many strings that were not even valid or syntactically correct. This prompted a new wave of research seeking to remedy this problem. Some work has been been based around *learning* a validity checker during training directly (Guimaraes et al., 2017; Janz et al., 2018). A larger branch of work has instead looked at using different output molecular representations because, much like when choosing how to input molecules (see Chapter 2), this output representation can have a large effect on how well the model performs. Examples to force models to close the branches and rings present in SMILES, include methods generating parse trees (Dai et al., 2018; Kusner et al., 2017), such as the grammar VAE (GVAE). There has also been work developing alternative text representations, which no longer have the same long-range dependencies that trip up models generating SMILES strings (Krenn et al., 2020; O'Boyle and Dalke, 2018).

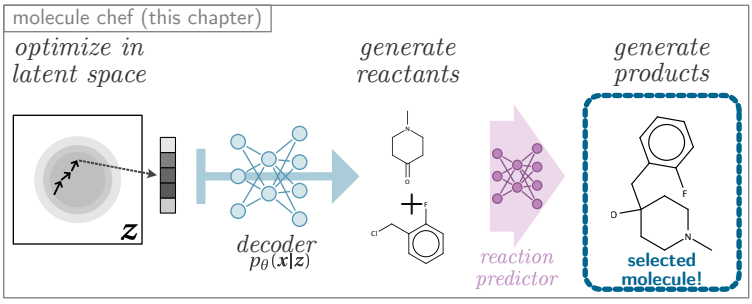


Figure 4.2 *An overview of our approach,* MOLECULE CHEF. MOLECULE CHEF *explicitly considers synthesizability as part of the design process. Instead of following previous work generating molecules directly (see Figure 4.1),* MOLECULE CHEF *first generates a "bag" of reactant molecules (chosen from a pool of easily obtainable building blocks), before reacting these molecules together to form a final (product) molecule. This process better reflects how molecules are made in practice.*

Elsewhere, there has been a series of investigations looking at using graph representations (Jin et al., 2018; Kajino, 2019; Li et al., 2018a; Liu et al., 2018; Samanta et al., 2019; Seff et al., 2019; Simonovsky and Komodakis, 2018; see also Box 4.3). Working with the graph representation of a molecule has several advantages. It avoids many of the problems with generating SMILES strings to start with. For example, a model cannot "forget" to close a bracket, "(", when generating an adjacency matrix. Furthermore, it also offers a more natural environment to impose or encourage other chemical concepts, such as valency constraints (e.g. a "C" atom cannot have more than four bonds; Liu et al., 2018).

The graph representation also facilitates the use of higher-level character sets. This means that instead of generating a molecule atom by atom, one can do it subgraph by subgraph. For example, the junction tree VAE (JT-VAE; Jin et al., 2018) forms whole rings at once, and later developments of this idea have used vocabularies consisting of even larger pre-defined fragments (Jin et al., 2020a; Podda et al., 2020). Using fragments means that we are generating less of the graph from scratch, with more of an onus on the model for working out how larger existing subgraphs should connect together. We can relate the use of fragments back to the trade-off between chemical coverage and viability, which we discussed before. Using pre-defined fragments, for example generated from the training data, can help generate stable molecules; yet this fragment vocabulary is also limiting, restricting the space of molecules we could feasibly explore.

4.1.3 Synthesizability

In the generative models for designing molecules we have discussed so far, there is no explicit indication that the designed molecules can actually be *synthesized*, the third tenet in our notion of viability. Being able to synthesize a molecule, the next step after designing

Box 4.3: Other categorizations of ML-based molecule design

In the main text we mainly focused on differentiating ML-based approaches by the *representation* they used for outputting molecules. An alternative method to categorize ML approaches is by model type or optimization technique (see e.g. Sanchez-Lengeling and Aspuru-Guzik, 2018, Fig. 4, or Schwalbe-Koda and Gómez-Bombarelli, 2020, Fig. 5). There have been a wide variety of ML models developed to produce molecules including generative adversarial networks (GANs; De Cao and Kipf, 2018), normalizing flows (Madhawa et al., 2019), autoregressive models (Li et al., 2018a,b; Mercado et al., 2020; Segler et al., 2018a), and autoencoders (Gómez-Bombarelli et al., 2018; Samanta et al., 2019; Simm and Hernández-Lobato, 2020). There has also been a wide variety of ML optimization techniques used to find better molecules, such as approaches based on reinforcement learning (RL; Neil et al., 2018; Olivecrona et al., 2017; Simm et al., 2020; Zhou et al., 2019), Bayesian optimization (Gómez-Bombarelli et al., 2018; Korovina et al., 2020), substructure identification and "stitching" (Jin et al., 2020b), and even the technique of treating optimization as a translation problem, from a bad molecule to a good molecule (Jin et al., 2020a, 2019).

We drew more attention to the output representation in the main text because often the same high-level approach can be used in different ML paradigms. For instance, the next chapter develops a model, DoG-Gen, that can be used both (a) as an autoregressive model and optimized using a RL routine, or (b) as the decoder in a larger autoencoder model. Likewise, within the same ML paradigm there is often work using different representations. For example, RL methods have been developed that construct molecules up through SMILES characters (Olivecrona et al., 2017; Segler et al., 2018a), an atom-by-atom/bond-by-bond approach (You et al., 2018a; Zhou et al., 2019), and even more recently through virtual reaction schemes similar to the methods proposed here (Gottipati et al., 2020; Horwood and Noutahi, 2020; see § 5.5.1).

it in the design-make-test-analyze cycle, is a prerequisite of any experimental testing. As such, this hampers the application of computer-designed compounds in practice, where rapid experimental feedback is crucial. While it is somewhat possible to address this in a post hoc manner using synthesis planning (see e.g. Cook et al., 2012; Gao and Coley, 2020; Segler et al., 2018b; Szymkuć et al., 2016), unfortunately this is very slow.

To address this concern, we must address the molecular recipe problem: what new molecules are we able to concoct, given a set of readily available building block molecules? In this chapter, inspired by older chemoinformatics work (see Box 4.1), we tackle this problem by developing a generative model, which we call Molecule Chef, that explicitly

includes synthesis instructions in the generative process. MOLECULE CHEF is a latent variable model than can generate new molecules through a two-part map (see Figure 4.2): First, a mapping from a continuous latent space to a subset of known and easy-to-obtain building blocks (to use as reactants). Second, a mapping from this set of selected building blocks to a final product molecule, based on a reaction predictor model (Chapter 3).

By including the idea of synthesis overtly in the ML-driven design of molecules, we argue that MOLECULE CHEF can better link up the design step and the make step in molecule search, and ultimately speed up the entire process. Furthermore, we argue that building up molecules through reactions, in effect selecting and then editing reactant molecules, provides a sensible inductive bias for viability, producing molecules that are valid, stable, and synthesizable. In the subsequent sections of this chapter we shall describe and evaluate our model.

4.2 MOLECULE CHEF: generating reactant bags

In this section we describe the architecture of MOLECULE CHEF and how we train it. MOLECULE CHEF consists of two core parts: (a) an autoencoder (see Box 4.2), acting as a co-occurrence model over a set of building block molecules (i.e. possible reactants), and (b) a reaction "oracle" that transforms the reactants chosen into a final product molecule. In the first subsection below we explain these two parts in more detail[2], before then going on to discuss how we train our model. By moving around the latent space of MOLECULE CHEF we can select different final product molecules as well as their "recipe", i.e. possible reactants to produce them.

Notation Before describing the architecture of MOLECULE CHEF in more detail, it is worth laying out some general notation. We define the set of all possible valid molecular graphs as $\mathcal{S}$, and we continue to denote an individual molecule in this space as $\mathcal{M} \in \mathcal{S}$. We also continue to assume we have access to representations of these molecules, $\boldsymbol{m}_{\mathcal{M}}$, computed using graph neural networks (GNNs, see also § 2.2.2). The set of common building block molecules, easily procurable by a chemist, and which we want to choose particular reactants from, we denote by $\mathcal{B} \subset \mathcal{S}$.

[2]Further details of the hyperparameters and architectures we use can be found in Appendix C.2.

4.2.1 The MOLECULE CHEF model

In this section we describe the architecture of MOLECULE CHEF. As discussed previously, MOLECULE CHEF consists of the composition of two parts: (a) an autoencoder that maps between a continuous latent space, $\boldsymbol{z} \in \mathbb{R}^d$, and a bag (i.e. multiset[3]) of easily procurable reactant molecules, $\boldsymbol{x} \subset \mathcal{B}$; and (b) a reaction "oracle", $\mathsf{Product}(\cdot)$, that transforms this bag of selected reactant molecules into a multiset of product molecules, $\boldsymbol{y} \subset \mathcal{S}$.

The benefit of this approach is that for part (b), the reaction oracle, we can pick from one of the several existing reaction predictor models that we reviewed last chapter. In this chapter we use the Molecular Transformer (MT) of Schwaller et al. (2019), because of its state-of-the-art performance[4] (see § 3.4.4; we also provide a short commentary on this design decision in Appendix C.2.3).

This leaves us with the task of defining part (a), an autoencoder over a bag of reactants. We can further break this down into two components: (i) the *encoder*, $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, that maps from the bag of reactants to latent space; and (ii) the *decoder*, $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, that maps back from the latent space to the bag of reactants. Below we shall describe each of these components in turn, before describing how we train the resulting model in the next subsection.

Encoder

The structure of MOLECULE CHEF's encoder, $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, is shown in Figure 4.3. For each data point the encoder has as input the multiset of reactant molecules chosen $\boldsymbol{x} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots\}$. It first computes a molecule-level embedding of each individual reactant molecule using a GNN (§ 2.2.2), before summing these embeddings to get a representation that is invariant to the order of the multiset. A feedforward neural network is then used to parameterize the mean and variance of a Gaussian distribution over $\mathbf{z}$.

Decoder

The decoder, $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, maps back from the latent space to a multiset of reactant molecules. These reactants are typically small molecules, which means we could fit a deep generative model which produces them from scratch. However, to better mimic the more realistic

[3]Note how (by using a *multiset*) we allow molecules to be present multiple times as reactants in our reaction, although in practice many reactions only have one instance of a particular reactant.

[4]Although the Molecular Transformer achieves excellent performance on benchmarks (Schwaller et al., 2019), it is worth pointing out that none of the current reaction predictor models we described in the previous chapter are perfect, and they will sometimes predict incorrect reactions. However, by treating this oracle as a black box, our model can take advantage of any future developments of these methods or use alternative predictors for other reasons, such as to control the reaction classes used.

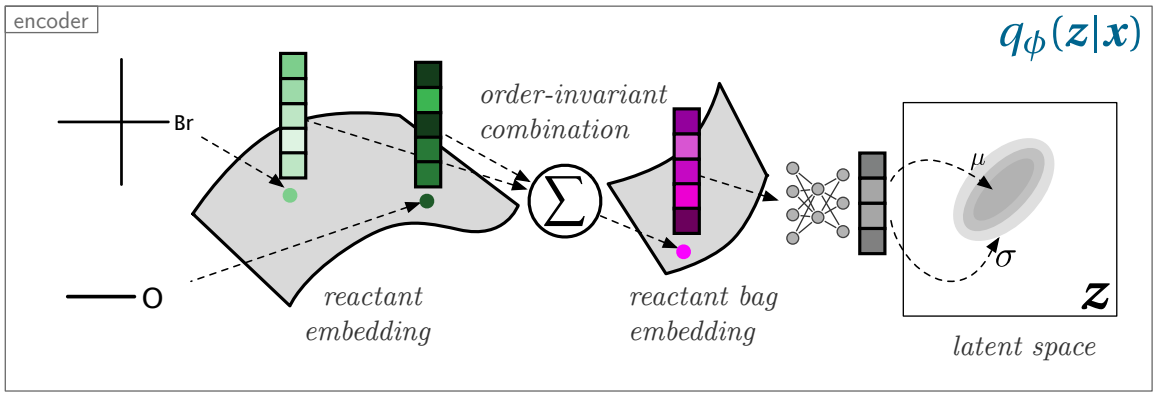


Figure 4.3 *The encoder of* MOLECULE CHEF. *This maps from a multiset of reactants to a distribution over latent space. There are three main steps: (i) the reactants molecules are embedded into a continuous space using GNNs (see § 2.2.2) to form molecule-level representations; (ii) the molecule-level representations in the multiset are summed to form one order-invariant representation for the whole multiset; (iii) this is then used as input to a feedforward neural network which parameterizes a Gaussian distribution over* $\boldsymbol{z}$.

process of selecting molecules from an easily obtainable, fixed set of possible building blocks, $\mathcal{B}$, we instead restrict the output of the decoder to pick the reactants from this set, i.e. $\boldsymbol{x} \subset \mathcal{B}$.

This happens in a sequential process using a recurrent neural network (RNN), with the full process described in Algorithm 4.1 (and shown at a high-level by Figure 4.4). The latent vector, $\boldsymbol{z}$, is used to parameterize the initial hidden layer of the RNN. The reactants chosen are fed back in as inputs to the RNN at the next generation stage. While training the decoder we use teacher forcing and randomly sample the order of the reactants.

4.2.2 Training objective

When training MOLECULE CHEF, we train the autoencoder and the reaction predictor separately. The reaction predictor is trained independently on a reaction dataset. For the autoencoder we can train the encoder and decoder in a purely *unsupervised* way using a WAE objective (see Box 4.2), although alternative autoencoder objectives would also be suitable here.

Predicting properties from the latent space Sometimes we might have labels associated with each reactant bag, which we might wish to use in a *supervised* way to shape the latent space. For example, each final product, $\boldsymbol{y}$, might be associated with a desirable property, $\boldsymbol{w}$, for which we later wish to use MOLECULE CHEF to optimize for. Therefore, in a similar manner to Liu et al. (2018, § 4.3) and Jin et al. (2018, § 3.3), we can train a

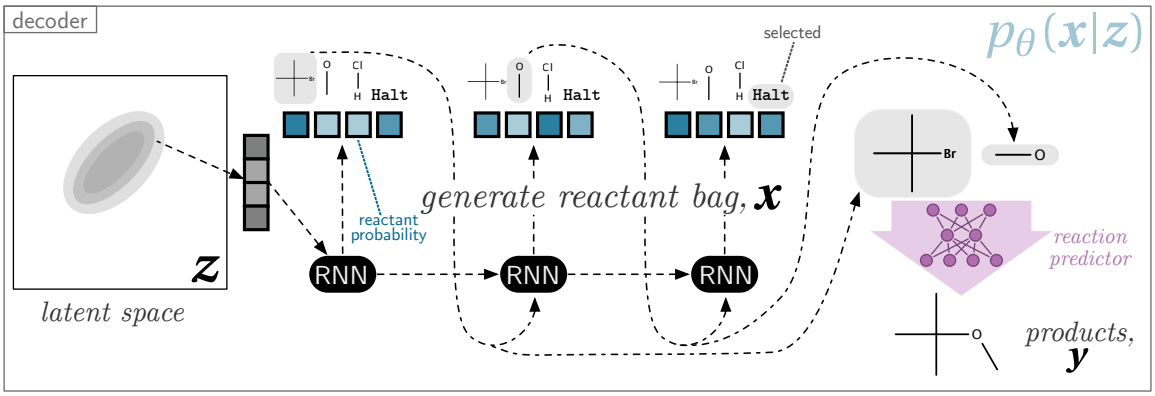


Figure 4.4 *The decoder of* MOLECULE CHEF *(see Alg. 4.1 for further details). The decoder generates a bag (i.e. multiset) of reactants in sequence through calls to a RNN. At each step the model picks either one molecule from the pool of all possible building blocks, $\mathcal{B}$, or to halt, finishing the sequence. The latent vector, $\boldsymbol{z}$, is used to parameterize the initial hidden layer of the RNN. Reactants that are selected are fed back into the RNN on the next step. The reactant bag formed is later fed through a reaction predictor to form a final product. We represent molecules in this process using molecule-level embeddings computed using the same GNN (including the same weights) as the one used in the encoder.*

Algorithm 4.1: MOLECULE CHEF*'s Decoder*

Input:
- Latent space sample, $\boldsymbol{z}$;
- GNN to create molecule embeddings, $\mathsf{GNN}(\cdot)$;
- RNN (recurrent neural network), $\mathsf{RNN}(\cdot,\cdot)$;
- Set of available (e.g. cheap, easy-to-obtain, etc.) building blocks, $\mathcal{B}$, to use as possible reactants;
- A "halt" embedding (part of MOLECULE CHEF's parameters), $\boldsymbol{h}_{\not\vdash}$;
- A linear projection from latent space to initialize the initial hidden vector of the RNN (part of MOLECULE CHEF's parameters), $\mathsf{Lin}(\cdot)$;
- A maximum size on the reactant bag, $S^{\max}$.

Result:
- Set of reactants, $\boldsymbol{x} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots\}$.

/* Initialize first inputs into RNN: */

$\boldsymbol{h}_0 \leftarrow \mathsf{Lin}(\boldsymbol{z})$ ▷ Initialize hidden layer of RNN using latent variable.

$\boldsymbol{m}_0 \leftarrow \boldsymbol{0}$ ▷ Start of sequence symbol.

/* We now sequentially predict these outputs using the RNN: */

for s *in* $[1, \ldots, S^{max}]$ **do**

  $\boldsymbol{h}_s \leftarrow \mathsf{RNN}(\boldsymbol{h}_{s-1}, \boldsymbol{m}_{s-1})$ ▷ Update RNN hidden vector.

  $\boldsymbol{B} \leftarrow \mathsf{STACK}\left([\mathsf{GNN}(\mathcal{M}) \text{ for all } \mathcal{M} \text{ in } \mathcal{B}] \parallel [\boldsymbol{h}_{\not\vdash}]\right)$ ▷ Form representations for all possible actions.

  $\boldsymbol{l} \leftarrow \boldsymbol{h}_s \boldsymbol{B}^t$ ▷ Form logits from dot products ($\cdot^t$ is transpose).

  $\mathcal{M}_s \sim \mathsf{softmax}(\boldsymbol{l})$ ▷ Sample reactant/halt token from categorical distribution.

  if $\mathcal{M}_s = \not\vdash$ **then**

    break ▷ If sampled halt token then exit loop.

  else

    $\boldsymbol{m}_s \leftarrow \mathsf{GNN}(\mathcal{M}_s)$ ▷ Else form representation of molecule chosen for RNN's next input.

return $\{\mathcal{M}_1, \mathcal{M}_2, \ldots\}$

two hidden-layer property predictor neural network. This network tries to predict the property, $\boldsymbol{w}$, of the final product, $\boldsymbol{y}$, from the latent encoding of the associated bag of reactants. We add the loss from this predictor to our WAE objective and train this property predictor network simultaneously with the encoder and decoder. As well as shaping the latent space, we can also make use of this property predictor, after it has been trained, to perform local optimization; this is investigated in Section 4.3.2.

4.3 Evaluating MOLECULE CHEF

In this section we evaluate MOLECULE CHEF. In particular, we look at (i) its ability to generate a diverse set of viable molecules; (ii) how useful its learned latent space is for local optimization (to find product molecules exhibiting some property); and (iii) whether by training a separate regression model back from product molecules to the latent space, MOLECULE CHEF can be used as part of a setup to perform retrosynthesis.

Data In order to train our model we need a dataset of reactant bags. For this, similar to the last chapter (§ 3.3.1), we use the USPTO reaction dataset (Lowe, 2012). We again use the same processed and cleaned version from Jin et al. (2017), and follow the same filtering out of reagents as we did last chapter. However, given that the Molecular Transformer can also work with non LEF (linear electron flow, see § 3.1.3) reactions, we no longer remove these reactions from our training dataset.

We wish to define as building block molecules only popular molecules that a chemist would have easy access to. To this end, we filter our training dataset (using again the train/validation/test splits from Jin et al., 2017) so that we keep only reactions containing reactants that occur at least 15 times across different reactions in the original larger USPTO training dataset. This leaves us with a dataset of 34,426 unique reactant bags for training MOLECULE CHEF, selected from 4,344 unique building block molecules.

When evaluating MOLECULE CHEF for generating molecules, we compare against baselines that generate molecules directly, through producing a molecular graph or SMILES string. For training these baselines, we combine the 4,344 unique building block molecules and all of the associated products from their recorded different combinations, as even though our autoencoder has not seen the products during training, the reaction predictor has.

Table 4.1 *Table showing the validity, uniqueness, novelty, and normalized quality (all as %, higher better) of the molecules generated from decoding from 20k random samples from the prior $\boldsymbol{z}$. Quality is the proportion of valid molecules that pass the quality filters proposed in Brown et al. (2019, § 3.3), normalized such that the score on the training set is 100. FCD is the Fréchet ChemNet Distance (Preuer et al., 2018), capturing a notion of distance between the distribution of generated valid molecules and the training dataset distribution (lower FCD scores are better). The uniqueness and novelty figures are also conditioned on validity.*

Model Name	Validity (↑)	Uniqueness (↑)	Novelty (↑)	Quality (↑)	FCD (↓)
AAE (Kadurin et al., 2017; Polykovskiy et al., 2020)	85.86	98.54	93.37	94.89	1.12
CGVAE (Liu et al., 2018)	100.00	93.51	95.88	44.45	11.73
CVAE (Gómez-Bombarelli et al., 2018)	12.02	56.28	85.65	52.86	37.65
GVAE (Kusner et al., 2017)	12.91	70.06	87.88	46.87	29.32
SMILES LSTM (Segler et al., 2018a)	91.18	93.42	74.03	100.12	0.43
MOLECULE CHEF	99.05	95.95	89.11	95.30	0.73

4.3.1 Generation

To assess MOLECULE CHEF's ability to generate new molecules, we perform both quantitative and qualitative evaluations. We want to ensure that MOLECULE CHEF can generate a wide range of new and viable molecules, particularly compared to previous ML methods which do not consider synthesizability constraints.

Quantitative evaluation We begin by analyzing our model using the metrics favored by previous work[5] (Kusner et al., 2017; Liu et al., 2018): validity, uniqueness, and novelty. We view these metrics as a way to obtain a useful and cheap validation of an approach (showing that sensible molecules are produced), albeit with limitations (see § 5.5.3).

Validity is defined as requiring that at least one of the molecules in the bag of products can be parsed by RDKit (RDKit Team, 2019). For a bag of products to be unique, we require it to have at least one valid molecule that the model has not generated before in any of the previously seen bags. Finally, for computing novelty, we require that the valid molecules not be present in the same dataset we use for training the baseline generative models.

In addition, we compute the Fréchet ChemNet Distance (FCD; Preuer et al., 2018) between the valid molecules generated by each method and our baseline training set. Finally, in order to try to further assess the "viability" of the molecules generated, we

[5] Note that we have extended the definition of these metrics to a bag (multiset) of products, given that our model can output multiple molecules for each reaction. However, when sampling 20,000 times from the prior of our model, we generate single product bags 97% of the time. This means in practice that we often end up using the same definition for these metrics as the previous work (which always generated single molecules).

record the (train-normalized) proportion of valid molecules that pass the quality filters proposed by Brown et al. (2019, § 3.3); these filters aim to remove molecules that are "potentially unstable, reactive, laborious to synthesize, or simply unpleasant to the eye of medicinal chemists".

For the baselines, we consider the character VAE (CVAE; Gómez-Bombarelli et al., 2018), the grammar VAE (GVAE; Kusner et al., 2017), the adversarial autoencoder (AAE; Kadurin et al., 2017), the constrained graph VAE (CGVAE; Liu et al., 2018), and an autoregressive model on SMILES using LSTMs with no latent space (Segler et al., 2018a). These models cover a wide range of approaches for modeling structured molecular graphs; however, they do not provide synthetic routes with the output molecule. Further details about the baselines can be found in Appendix C.2.2.

The results are shown in Table 4.1. As MOLECULE CHEF decodes to a bag made up from a predefined set of molecules, those reactants going into the reaction predictor are all valid. The validity of the final product is not 100%, as the reaction predictor can make non-valid edits to these molecules, but we see that in a high proportion of cases the products are valid too. Furthermore, what is very encouraging is that the molecules generated often pass the quality filters, giving evidence that the process of building molecules up by combining sensible reactant building blocks often leads to stable products.

Qualitative evaluation While we can somewhat quantify more subtle aspects of the viability of the generated molecules using the quality filters discussed above, these are manually curated rules and, as such, do not capture everything. As an alternative, we also showed some of the individual molecules generated to a domain expert.

To be more specific, to compare the CVAE, GVAE, and MOLECULE CHEF models, we first generated a set of nearby molecules using a random walk in latent space, starting from the encoding of a particular molecule (for MOLECULE CHEF we encode the reactant bag known to generate the same molecule). We showed the generated molecules to the domain expert and asked them to evaluate their properties in terms of their "stability", "toxicity", "oxidizing power", and "corrosiveness" (the rationales are provided in more detail in Appendix C.3). The results are shown in Figure 4.5; here we see that molecules produced by MOLECULE CHEF often look sensible, whereas molecules produced from the older CVAE and GVAE models often exhibit undesirable features.

4.3.2 Local optimization

We now turn to evaluate MOLECULE CHEF in its ability to find molecules exhibiting desirable properties. Following Gómez-Bombarelli et al. (2018), we look at optimizing

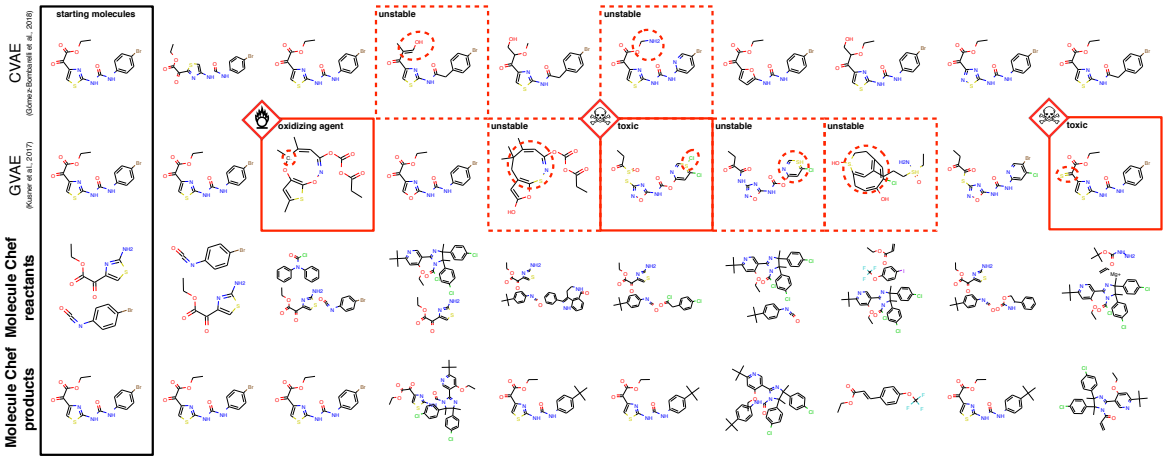


Figure 4.5 *Random walk in latent space. For each model we start from an encoding of the molecule in the left-most column (for* MOLECULE CHEF *we use the encoding of the associated reactants). We then walk randomly in the latent space until we decode to different outputs. (See Figure C.2 in the appendix for a blown up version of this figure.)*

molecules for their QED (Quantitative Estimate of Drug-likeness; Bickerton et al., 2012) score. While it has its shortcomings (Brown et al., 2019, p. 1106), the QED score offers a practical way to perform a proof of concept, as a cheap-to-evaluate mapping from the final product molecule to this score, $\boldsymbol{y} \mapsto \boldsymbol{w}$, exists in RDKit (RDKit Team, 2019).

Therefore, following the approach initially laid out in Section 4.2.2, when training MOLECULE CHEF we simultaneously train a property predictor network, mapping from the latent space of MOLECULE CHEF to the QED score of the final product. As well as organizing the latent space of the autoencoder during training, this network's gradients can be used at optimization time to do local optimization; below we consider such a procedure.

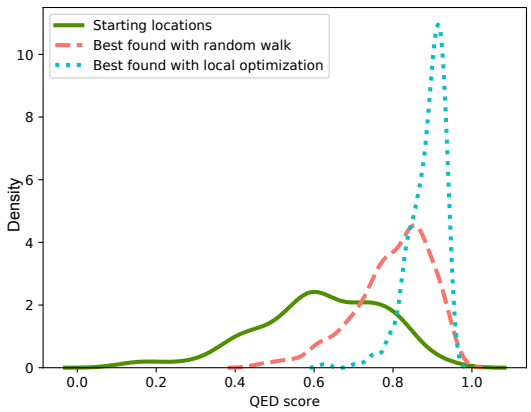


Figure 4.6 *KDE (kernel density estimate) plot showing that the distribution of the best QEDs found through local optimization (using our trained property predictor for QEDs) has higher mass over higher QED scores compared to the best found from a random walk. Note that the starting locations (green distribution) are sampled from the training data. The final products are predicted from the reactant bags using the Molecular Transformer.*

To be more specific, we evaluate the local optimization of molecular properties as follows. First, we take 250 bags of reactants and encode them into the latent space of MOLECULE CHEF. Secondly, we iteratively take small steps in latent space, in the direction of the gradient from the property predictor, until we have decoded to ten different reactant bags.

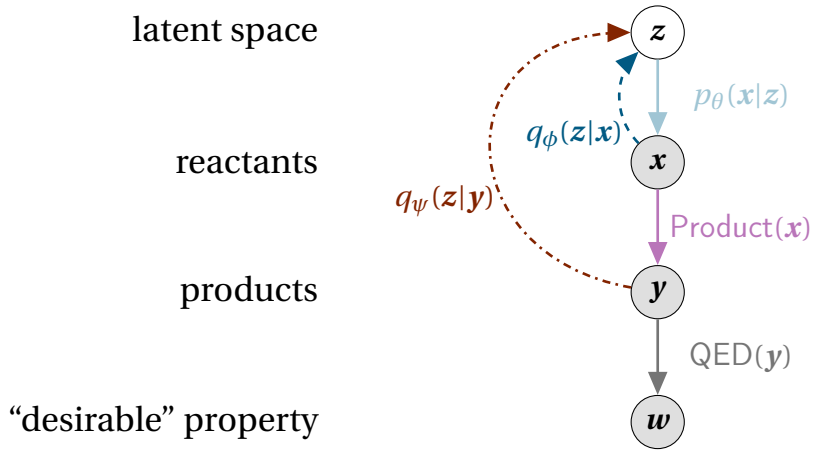


Figure 4.7 *Having learned a latent space which can map to products through reactants, we can learn a separate regression model, $q_\psi(z|y)$, back from the suggested products to the latent space (red dashed $- \cdot -$ arrow shown). We can then couple this with* MOLECULE CHEF*'s decoder, $p_\theta(x|z)$, to see if we can do retrosynthesis – the act of computing the reactants that create a particular product. Note that we train $q_\psi(z|y)$ after the parameters for the rest of the model have been learned.*

As a comparison we consider instead moving around latent space in a random walk (i.e. taking the same small steps as before but in random directions) until we have also decoded to ten different reactant bags. In Figure 4.6 we look at the distribution of the best QED score found in each run (considering the products from all ten reactant bags). As a comparison, we also plot the distribution of the initial QED scores.

When looking at individual optimization runs, we see that the QED scores vary considerably between different products, even if made with similar reactants. However, overall, Figure 4.6 shows that the distribution of the best QED score found is improved when purposefully optimizing for this. This is encouraging, as it gives evidence of the utility of our model in searching for better molecules.

4.3.3 Retrosynthesis

A unique feature of our approach is that we learn a decoder that maps from a latent representation to a bag of reactants. This gives us the ability to do retrosynthesis using the process shown in Figure 4.7. Specifically, we can train a separate regression model, which we denote as $q_\psi(z|y)$, to map from products to their associated reactants' representation in latent space. We can then compose this model with MOLECULE CHEF's decoder to get a function that maps all the way from products to reactants, thereby performing retrosynthesis (i.e. suggesting the reactants that produced a given product).

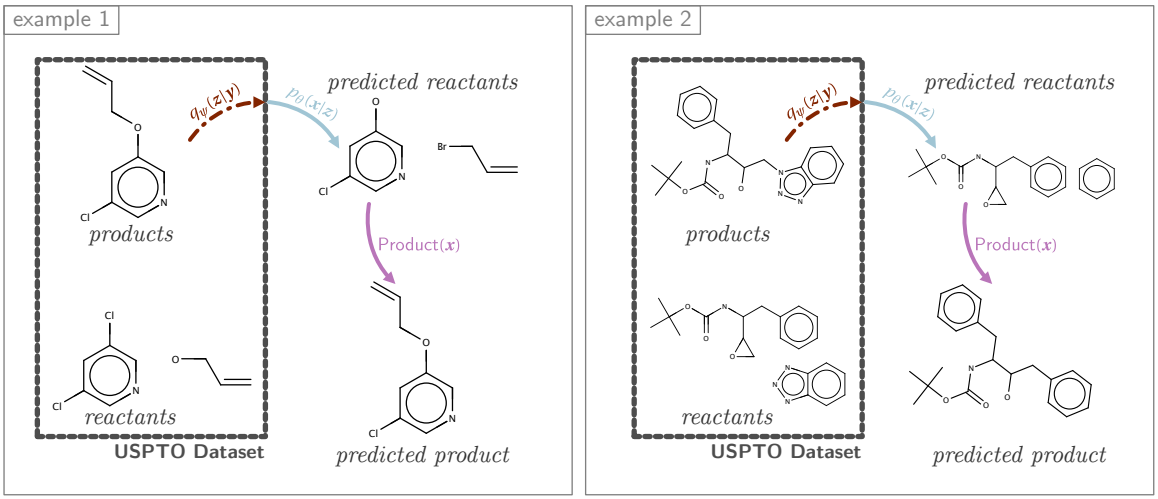


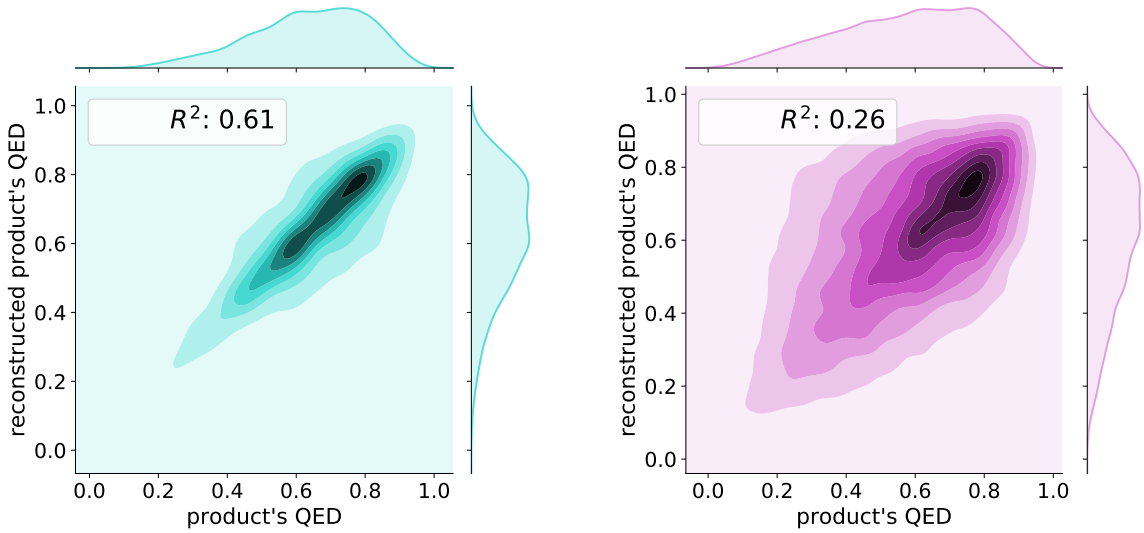
Figure 4.8 *An example of performing retrosynthesis prediction using a separate trained regression model from products to latent space. The "correct" reactants (i.e. the ones listed in the USPTO dataset) for the product molecule are in our list of building blocks.*

To implement this idea in practice, we train a small neural network for $q_\psi(\boldsymbol{z}|\boldsymbol{y})$, based on the same GNN architecture used in MOLECULE CHEF followed by four fully connected layers. For the training data, we use the set of product molecules in the USPTO data along with their associated reactants' location in an already trained MOLECULE CHEF's latent space[6]. Two examples of the predicted reactants from following this process are shown in Figure 4.8.

It is worth pointing out here that retrosynthesis is a difficult task; current state-of-the-art approaches for this task (a) work in a top-down manner, (b) exploit knowledge contained in very large reactant databases, (c) consider multiple steps, and (d) also have the ability to adjust already made decisions on seeing new data (Segler et al., 2018b). While we believe that our new method is unlikely to be competitive with complex planning tools, we believe it could open up new interesting and exciting approaches to this task, where perhaps a combination of bottom-up and top-down methods can be combined (see § 6.2.2).

Furthermore, it also allows us to pose and solve new problems. For instance, often we are not interested in reconstructing a target molecule per se, but obtaining a molecule with the target molecule's expected properties. Put another way, if faced with a hard or impossible to synthesize molecule, it would be valuable to be able to "project" this molecule back to synthesizable space, and be pointed directly to molecules with similar properties to the original molecule but that are easier to make.

[6] i.e. we are conditioning on fixed parameters ϕ.



(a) *Reachable products* **(b)** *Unreachable products*

Figure 4.10 *Kernel density estimate (KDE) plot (darker is higher density) assessing the correlation between the QED scores for the original product and its reconstruction (formed by performing the retrosynthesis process outlined in the main text and then putting these reactants through a reaction predictor). We assess on two portions of the test set; products that are produced from reactants in* MOLECULE CHEF*'s building block vocabulary are called "reachable products" (i.e. we know that they could be reconstructed), those that have at least one reactant that is absent are called "unreachable products".*

With this in mind, we assess our approach in the following way: (i) we take a product and perform retrosynthesis on it to produce a bag of reactants, (ii) we transform this bag of reactants using the Molecular Transformer to produce a new *reconstructed* product, and then finally (iii) we plot the resulting reconstructed product molecule's QED score against the QED score of the initial product. We evaluate on a filtered version of Jin et al. (2017)'s test set split of USPTO, where we have filtered out any reactions which have the exact same reactant and product multisets as a reaction present in the set used to train MOLECULE CHEF. In addition, we further split this filtered set into two distinct subsets: (a) "reachable products", which are reactions in the test set that contain as reactants only molecules that are in MOLECULE CHEF's vocabulary of possible building blocks, and (b) "unreachable products", which have at least one reactant molecule that is not in the list of available building blocks.

The results are shown in Figure 4.10; overall, we see that there is some correlation between the properties of products and the properties of their reconstructions. This is more prominent for the reachable products, which we believe is because our latent space is only trained on reachable product reactions and so is better able to model these.

Furthermore, some of the unreachable products may also require reactants that are not available in our pool of easily available building blocks, at least when considering one-step reactions. However, given that unreachable products have at least one reactant which is not in MOLECULE CHEF's vocabulary, we think it is very encouraging that there still is some, albeit smaller, correlation with the original molecule's QED. This is because it shows that our model can suggest molecules with similar properties made from reactants that are available.

4.4 Summary

In this chapter we introduced MOLECULE CHEF, a model able to construct *synthesizable* molecules by selecting reactants from a pool of easy-to-obtain building block molecules and then running them once through a reaction predictor. We demonstrated MOLECULE CHEF is able to generate a diverse range of molecules, and showed the utility of its latent space for both optimization and retrosynthesis tasks. In this manner, MOLECULE CHEF combines the strengths of modern VAE-based molecule models with those of older molecule design techniques: allowing us to organize and explore a large chemical space, while ensuring the molecules we suggest are viable. The next chapter builds on this work, exploring how we can extend the ideas here to enable the generation of *multi-step* synthesis plans.

Chapter 5

Barking up the Right Tree: Generating Molecule Synthesis DAGs

In the previous chapter we introduced the trade-offs inherent in ML-driven molecule design; in particular, one has to balance the size of the chemical space a method considers with the viability of the molecules it generates. Key to viability is the notion of synthesizability, or in other words being able to actually make the molecules being suggested. We argued that this could be achieved by building the generative process around chemical reactions. However, in designing our model, MOLECULE CHEF, we restricted it to performing a single reaction only, whereas in reality many molecules may require synthesis plans consisting of multiple reaction steps (see Figure 5.1).

In this chapter, to address this gap, we extend the ideas of MOLECULE CHEF to *multi-step molecular synthesis routes* by introducing a model called DOG-GEN. DOG-GEN generates a synthesis directed acyclic graph (DAG), which explains how the same simple building blocks we used in the last chapter can *recursively* combine to form ever more complicated molecules of interest. This allows us to maintain the advantages of building the generative process around the inductive bias of using reactions, while also obtaining the ability to search over much larger chemical spaces, in effect turning an inherently constrained optimization problem into an unconstrained one.

The contributions of this chapter are as follows: (i) we explain how one can represent synthetic routes as directed acyclic graphs (DAGs); (ii) we propose an efficient serialization procedure for such DAGs, enabling us to model them autoregressively using DOG-GEN; and (iii) we demonstrate how one can use DOG-GEN as a component in larger ML architectures and frameworks, such as reinforcement learning–based optimization techniques, or even autoencoder-based models (leading to a variant of our general model called DOG-AE). We evaluate our approach on a series of generation and optimization tasks, where we

show we can get competitive results to previous unconstrained approaches, while also ensuring the synthetic tractability of the molecules we suggest. The chapter wraps up with a discussion of our approach, indicating possible future directions.

Looking backward, this chapter builds on the previous one by extending the ideas of MOLECULE CHEF to multiple steps of reactions. However, rather than solely focusing on autoencoders, we first design a more general autoregressive model of DAGs, DOG-GEN. We then explain how this can be used in different ways, including in an autoencoder, which we call DOG-AE. Building these models also takes advantage of the graph neural networks and reaction predictors introduced in Chapters 2 and 3.

5.1 Synthesis pathways as DAGs

We begin by explaining how we can represent any multi-step synthesis plan as a DAG (see Figure 5.1).

As an example, consider the synthesis DAG for paracetamol shown in Figure 5.1, which we will use as a running example to illustrate our approach. Here, to make the final molecule, in this case paracetamol ($\mathcal{M}_7$), we are required to perform a series of chemical reactions. Each reaction (shown by the green arrows) takes a set of reactants and transforms them into a product (outlined in green). These reactants can be chosen from a pool of molecules that are available at that point, with the pool consisting of a large set of initial, easy-to-obtain, and always available starting molecules (i.e. building blocks, shown in blue) and the intermediate products already created.

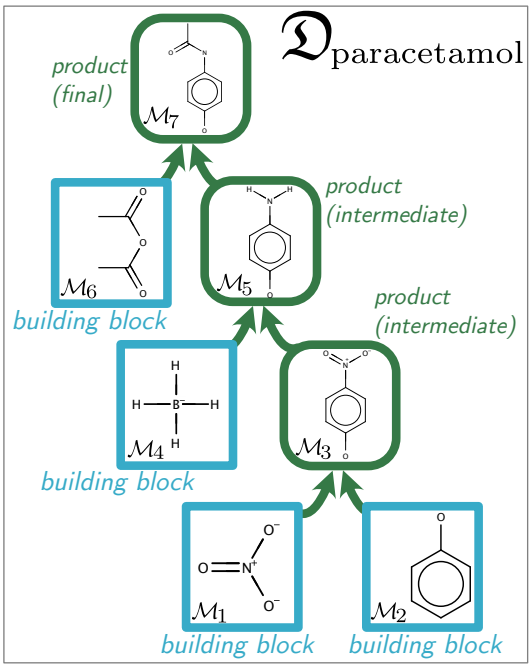
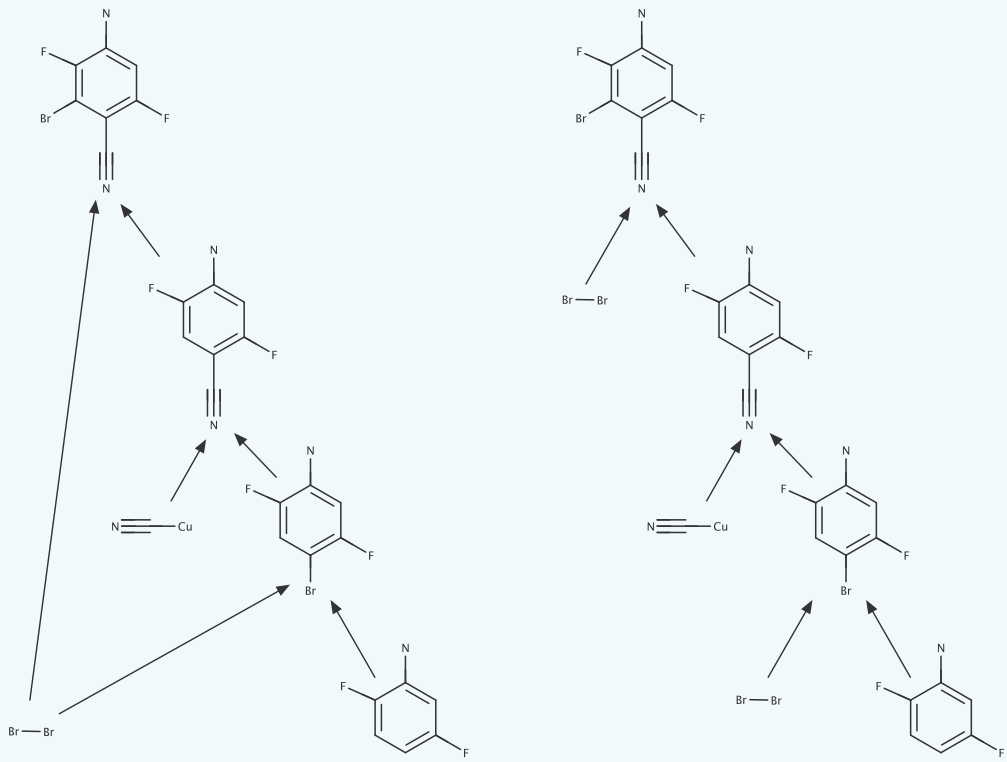


Figure 5.1 *An example synthesis DAG for paracetamol (Ellis, 2002). Note that we are ignoring some reagents, conditions and details of chirality for simplicity. The initial building blocks (shown in blue) combine in chemical reactions (shown by arrows) to form products (shown in green). These products can then become reactants themselves, and the process continues until the final molecule, in this case $\mathcal{M}_7$, is formed.*

We assume here that the reactions are all deterministic and produce a single primary product. And as such, we can represent any multistage reaction pathway using a synthesis DAG, which we shall denote $\mathfrak{D}$. Specifically, note that the DAG is directed

Box 5.1: Synthesis routes as DAGs

We specifically treat synthetic routes as "DAGs" (directed acyclic graphs) rather than as "trees". Below we show the difference between the two representations:



(a) *Representing synthetic route using* ***DAG*** *formalism (as used in this chapter).*

(b) *Representing synthetic route using* ***tree*** *formalism (note that generally in a tree one might draw the arrows from the final product node down; here we have drawn them upwards to indicate the forward direction of the reactions).*

In the DAG formalism there is a one-to-one mapping between each molecule and each node (see for example Br_2 in the above figure). In the tree formalism (a special form of directed acyclic graph where there is a unique path between any two nodes), we have to repeat reactant nodes that are used multiple times.

Given a complete synthetic route, there is little difference between the two representations; it is easy to convert between them by duplicating nodes or folding nodes into one another. The advantage of the DAG formalism comes at generation time. During generation when using the DAG formalism, we only need to generate each unique molecule once. This can be advantageous if a complex intermediate product, which requires several steps to create, needs to be used multiple times in the graph, as in the DAG formalism it only needs to be created once.

from reactants to products, and it is not cyclic, as we do not need to consider reactions that produce already obtained reactants. Also, note that in this representation there is a one-to-one mapping between each molecule and each node even if it occurs in multiple reactions (see Box 5.1). As each node in the DAG represents a molecular graph, we can describe the entire data structure as a synthesis DAG of (molecular) graphs (DoG).

5.2 The DoG-Gen model

Here we will explain our model over multi-step synthesis DAGs in more detail. We split this section into two, first describing how we can serialize a synthesis DAG into a sequence of construction actions, before describing our generative model, DoG-Gen, over such a serialized sequence. This will lay the foundation for the next section, in which we describe how we can use DoG-Gen in different ways.

5.2.1 Serializing the construction of DAGs

We need a way to "serialize" the construction of a DAG such that a ML model can iteratively construct it. Figure 5.3 shows such an approach. Specifically, we divide actions into three types: A1. *Node-addition (shown in yellow)*: what type of node (building block or product) should be added to the graph? A2. *Building block molecular identity (in blue)*: once a building block node is added, what molecule should this node represent? A3. *Connectivity choice (in green)*: what reactant nodes should be connected to a product node (i.e. what molecules should be reacted together)?

As shown in Figure 5.3 the construction of the DAG then happens through a sequence of these actions. Building block ('B') or product nodes ('P') are selected through action type A1, before the identity of the molecule they contain is selected. For building blocks this consists of choosing the relevant molecule through an action of type A2. Product nodes' molecular identity is instead defined by the reactants that produce them, therefore action type A3 is used repeatedly to either select these incoming reactant edges, or to decide to form an intermediate ($\not\rightarrow_{\mathcal{I}}$) or final product ($\not\rightarrow_{\mathcal{F}}$). In forming a final product all the previous nodes without successors are connected up to the final product node, and the sequence is complete.

In creating a DAG, $\mathfrak{D}$, we will formally denote this sequence of actions, which fully defines its structure, as $\mathfrak{D} = [V_1, V_2, V_3, \ldots, V_J]$. We shall denote the associated action types as $\mathsf{T} = [\mathsf{T}_1, \mathsf{T}_2, \mathsf{T}_3, \ldots, \mathsf{T}_J]$, with $\mathsf{T}_j \in \{\text{A1}, \text{A2}, \text{A3}\}$, and note that these are fully defined by the previous actions chosen (e.g. after choosing a building block identity you always go

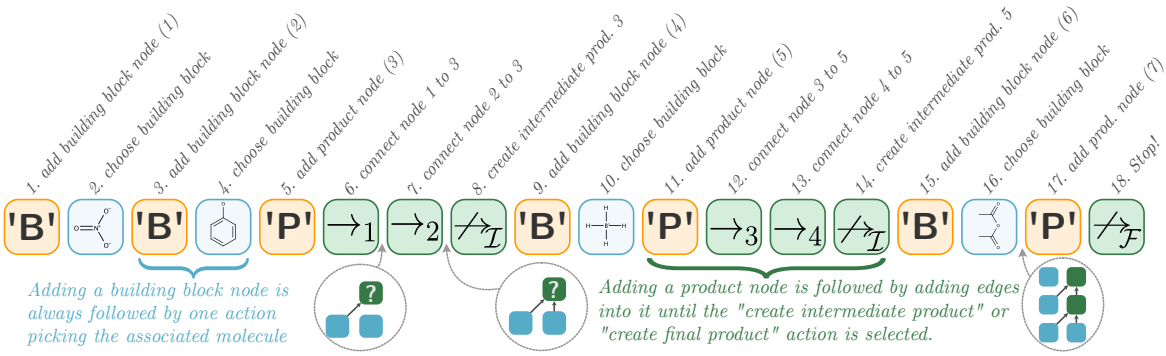


Figure 5.3 *An example of how we can "serialize" the construction of the DAG shown in Figure 5.1. The gray circles indicate the corresponding DAG existing at three different time-points in the construction sequence. The serialized construction sequence consists of a sequence of actions. These actions can be classified into three different types: (A1) node addition, (A2) building block molecular identity, and (A3) connectivity choice. By convention, we start at the building block node that is furthest from the final product node, sampling randomly when two nodes are at equivalent distances.*

back to adding a new node). The molecules corresponding to the nodes produced so far we will denote as $\mathcal{M}_i$, with i referencing the order in which they are created. We will also abuse this notation and use $\mathcal{M}_{<j}$ to describe the set of molecule nodes existing at the time of predicting action j. Finally, following the notation used in the last chapter for Molecule Chef, we shall continue to denote the pool of initial, easy-to-obtain building blocks as $\mathcal{B}$.

5.2.2 Defining a generative model over construction actions

We are now ready to define a generative model over $\mathfrak{D}$, called DoG-Gen. We allow our model to depend on a latent variable, $\boldsymbol{z} \in \mathbb{R}^d$. By default, we set this to a constant (such as the all zeros vector, $\mathbf{0}$), but we discuss alternatives in the parts that follow. We propose to model the DAG using an autoregressive factorization over the actions:

$$p_\theta(\mathfrak{D}|\boldsymbol{z}) = \prod_{j=1}^{J} p_\theta(V_j|V_{<j}, \boldsymbol{z}). \tag{5.1}$$

Each $p_\theta(V_j|V_{<j}, \boldsymbol{z})$ is parameterized by a neural network, with weights θ. The structure of this network is shown in Figure 5.4. It consists of a shared RNN module that computes a "context" vector. This "context" vector is then fed into a feedforward action-network for predicting each action. A specific action-network is used for each action type, and the action type also constrains the actions that can be chosen from this network: $V_j^{|\mathsf{A1}} \in \{\text{'B'}, \text{'P'}\}$, $V_j^{|\mathsf{A2}} \in \mathcal{B}$, and $V_j^{|\mathsf{A3}} \in \mathcal{M}_{<j} \cup \{\not\rightarrow_{\mathcal{I}}, \not\rightarrow_{\mathcal{F}}\}$. The hidden layer of the RNN is initialized

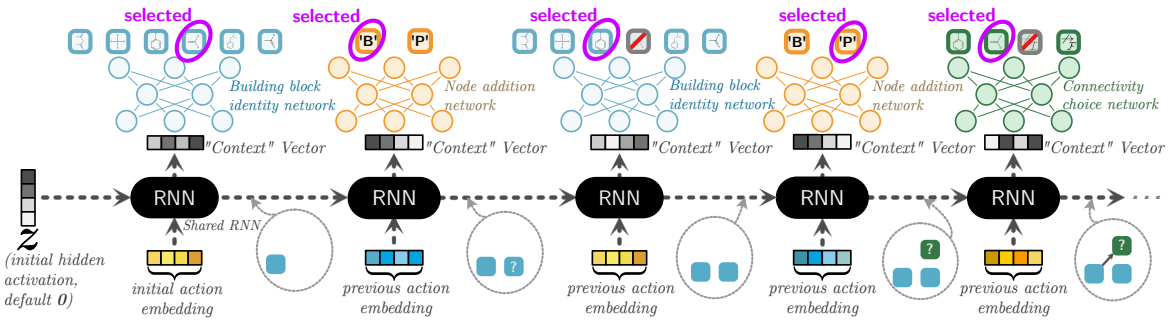


Figure 5.4 *A depiction of how we use neural networks to parameterize the probability of picking actions at stages 1-6 of Figure 5.3 (note that as stage 1 always suggests a building block node, it is automatically completed). A shared RNN for the different action networks receives an embedding of the previous action chosen and creates a context vector for the action network. When using our model as part of an autoencoder network, the initial hidden layer is parameterized by the latent space sample, $\boldsymbol{z}$. Each type of action network chooses a subsequent action to take (with actions that are impossible being masked out, such as selecting an already existing building block or creating an intermediate product before connecting up any reactants). The process continues until the "create final product" node is selected. Graph neural networks are used for computing embeddings of molecules where required. See Figure D.2 in Appendix D.2 for the subsequent steps of this procedure.*

by a linear transform of $\boldsymbol{z}$, and we feed in the previous chosen action embedding as input to the RNN at each step. Algorithm D.1, in Appendix D.2, contains pseudocode for the entire generative procedure.

Action embeddings For representing actions to our neural network we need continuous embeddings, $\boldsymbol{h}_V \in \mathbb{R}^d$. Actions can be grouped into either (i) those that *select a molecular graph* (be that a building block, $g \in \mathcal{B}$, or a reactant already created, $g' \in \mathcal{M}_{<j}$); or (ii) those that perform a more *abstract action* on the DAG, such as creating a new node ('B', 'P'), producing an intermediate product ($\not\rightarrow_{\mathcal{I}}$), or lastly producing a final product ($\not\rightarrow_{\mathcal{F}}$). For the abstract actions, the embeddings we use are each distinct learned vectors that are parameters of our model. With actions involving a molecular graph, following the previous chapters, we again use embeddings computed using GNNs (§ 2.2.2).

Reaction prediction When forming an intermediate or final product we need to obtain the molecule that is formed from the reactants chosen. At training time we can simply fill in the correct molecule using our training data. However at test time this information is not available. We assume however, as we did for MOLECULE CHEF, that we have access to an oracle, Product(·), which can perform reactions for us. Again we use a pre-trained Molecular Transformer (Schwaller et al., 2019) for this oracle in evaluating the models of this chapter. We take the top one prediction from the Transformer for the product, and if

this is not a valid molecule (determined by RDKit), then we assume no reaction occurred and instead pick one of the reactants randomly.

When running our model at prediction time there is the possibility of getting loops (and so no longer predicting a DAG). This can happen if the output of a reaction (either intermediate or final) creates a molecule which already exists in the DAG as a predecessor of one of the reactants. A principled approach one could use to deal with this when using a probabilistic reaction predictor model, such as the Molecular Transformer, is to mask out, in the reaction predictor's beam search, the prediction of reactions that cause loops. However, in our experiments we treat the reaction predictor as a black box oracle, for which we send reactants to and for which it sends us back a product. Therefore, to deal with any prediction-time loops we go back through the DAG, before and after predicting the final product node, and remove any loops we have created by choosing the first path that was predicted to each node.

5.3 DoG-Gen as a component in larger frameworks

Having introduced our basic generative model for generating synthesis DAGs of (molecular) graphs (DoGs), called DoG-Gen, here we introduce further use cases for our model. First, we describe how we can use the DoG-Gen model, as is, for performing molecular optimization via fine-tuning or reinforcement learning (RL). One can also view DoG-Gen as a general, learnable mapping from vectors to synthesis DAGs, and as such, one can use it as a component in larger machine learning models. To this end, we next develop an autoencoder called DoG-AE; DoG-AE is able to associate DAGs with continuous latent embeddings, which is useful for interpolating within synthesis DAG space.

5.3.1 Molecular optimization with DoG-Gen via fine-tuning

To use DoG-Gen to perform molecule optimization we adopt the hill climbing algorithm from Brown et al. (2019, § 7.4.6);Neil et al. (2018); Segler et al. (2018a). The process involved is described in Algorithm 5.1. It starts with a DoG-Gen model that has been pre-trained via maximum likelihood to match the training dataset distribution, $p_{\text{data}}(\mathfrak{D})$. The process then consists of repeating a series of fine-tuning steps. At each step, a large number of candidate DAGs are sampled from DoG-Gen and evaluated according to some target, before DoG-Gen's weights are then fine-tuned on the top K DAGs seen so far.

We can interpret this approach as an implementation of the cross-entropy method (de Boer et al., 2005). It can also be viewed as a variant of the REINFORCE (Williams,

Algorithm 5.1: *High-level overview of the DAG fine-tuning procedure. NB for* DOG-GEN, $\boldsymbol{z}$ *is always set at a constant (e.g.* $\mathbf{0}$*), hence we drop our specific dependence on* $\boldsymbol{z}$ *in this algorithm.*

Input:
- DOG-GEN model, $p_\theta(\mathfrak{D})$;
- Number of fine-tuning iterations, I^{iter};
- Number of samples, N^{samples};
- Threshold for number of samples to keep, K (typically $K < N^{\text{samples}}$);
- Objective function to optimize, $\text{obj}(\cdot)$.

Result:
- List of all synthesis DAGs and their score (ranked), Q;
- DOG-GEN model with updated weights, $p_\theta(\mathfrak{D})$.

```
Initialize an empty list to store the results, Q.
Compute the score, i.e. using obj(·), of all products in the initial training set and add to Q.
for i in [1,...,I^iter] do
    Sample N^samples from p_θ(𝔇).
    Compute the score, i.e. again using obj(·), of the final products of these sampled DAGs and add
      to list Q.
    Select the K DAGs with the highest score from Q.
    Run two training epochs on θ using these K DAGs as training data.
return Q and p_θ(𝔇)
```

1992) algorithm with a particular reward shaping. We could also use more complicated RL algorithms here. Nevertheless, we find empirically that this simple approach works well.

5.3.2 DOG-AE: learning a latent space over synthesis DAGs

It is often useful to learn representations of data in an unsupervised manner (see Box 4.2). For instance, we showed in the previous chapter that we could organize the complex space of reactant bags (and their associated products) using an autoencoder, later allowing us to sample and interpolate within this space. We now apply similar ideas to the modeling of molecular synthesis DAGs and develop a new autoencoder, DOG-AE. Here, each latent variable, $\boldsymbol{z}$, for which we assign a Gaussian prior, can be thought of as describing different types of multi-step synthesis pathways.

We will break the description of the architecture of DOG-AE down into the same topics as those we used to define MOLECULE CHEF; that is we shall go through the subjects of (a) the encoder, a stochastic mapping from synthesis DAGs to a continuous latent space; (b) the decoder, a stochastic mapping from latent space back to synthesis DAGs; and (c) the training objective. The decoder and the training objective are the easiest to explain. For the decoder, we use the already defined DOG-GEN model, with the latent variable, $\boldsymbol{z}$, initializing the hidden state of the RNN as shown in Figure 5.4 (rather than setting this

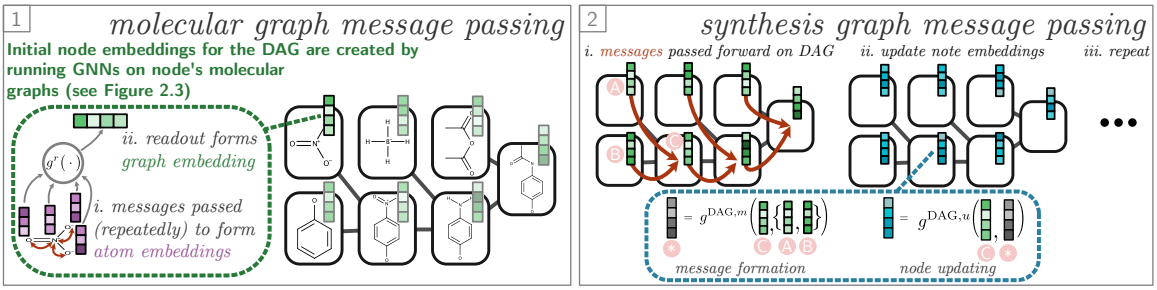


Figure 5.5 *The encoder for* DoG-AE *embeds the DAG of Graphs (DoG) into a continuous latent space. It does this using a two-step hierarchical message passing procedure. In step 1 (molecular graph message passing) it computes initial embeddings for the DAG nodes by forming graph-level embeddings using a GNN on the molecular graph associated with each node (see § 2.2.2). In step 2 (synthesis graph message passing), a message passing algorithm is again used; however, this time it is used on the synthesis DAG itself, passing messages forward. In our experiments we use GGNNs (Li et al., 2015) for both message passing steps (see Appendix D.4.1 for further details). The final representation of the DAG is taken from the node embedding of the final product node.*

to a constant). For the training objective, we use the same WAE loss that we used for Molecule Chef (see Box 4.2). This leaves us just needing to define the encoder.

Encoder At a high level, our encoder consists of a two-step hierarchical message passing procedure described in Figure 5.5: (i) molecular graph message passing, and (ii) synthesis graph message passing. As we have discussed before, each node in a synthesis DAG is itself a graph representing a molecule. For step (i) we obtain continuous embeddings for the molecules representing each node. This is done using the same molecule GNN used for representing molecule-associated actions in DoG-Gen (also using the same weights).

Step (ii) involves performing a second round of message passing, this time across the synthesis DAG. We initialize each node in the synthesis DAG with its associated molecule-level embedding (i.e. the graph-level embeddings for the molecules become the initial node-level embeddings for the DAG). We then update these embeddings using a separate GNN. The final node embeddings in the DAG are aggregated using the GNN's readout function to form a DAG-level representation which is used to parameterize a distribution over latent space.

Much like the GNN for computing the molecule-level embeddings in step (i), we are flexible about the exact GNN architecture we use to perform message passing across the DAG in step (ii). In the experiments later in this chapter, we shall again use a model based around the GGNN (Gated Graph Sequence Neural Network; Li et al., 2015; see also § 2.2.2) architecture but defer specifics to Appendix D.4.1.

5.4 Evaluation

We now turn to evaluating our approach on both generation and optimization tasks. To train our models, we again extract a suitable dataset out of the USPTO reaction dataset (§ 3.3.1). Starting with the same pool of 4,344 building blocks that we used for MOLECULE CHEF (§ 4.3), we compose reactions in the dataset together to form the DAGs (see Appendix D.1 for full details on how this is done). We train both our DOG-GEN and DOG-AE models on the same dataset and find that DOG-AE obtains a reconstruction accuracy (on our held out test set) of 65% when greedily decoding (i.e. picking the most probable action at each stage of decoding).

5.4.1 Generative modeling of synthesis DAGs

For evaluating how well our models can generate new molecules we start by following a similar setup to that used last chapter. Specifically, we focus on assessing the properties of the *final* product molecules in the synthesis DAGs. This allows us to compare to the same baselines and use the same metrics as the previous chapter (see § 4.3.1)[1]. Furthermore, we also include comparisons to the GraphVAE (Simonovsky and Komodakis, 2018) and the junction tree variational autoencoder (JT-VAE; Jin et al., 2018) models.

Table 5.1 shows the results. Generally, we see that many of these models perform comparably with no model performing better than all of the others on all of the tasks. The baselines JT-VAE and MOLECULE CHEF have relatively high performance across all the tasks, although by looking at the FCD score it seems that the molecules that they produce are not as close to the original training set as those suggested by the simpler character-based SMILES models, CVAE or SMILES LSTM. Encouragingly, we see that the models we propose achieve good performance on these tests and obtain scores comparable with many of the models that do not provide synthetic routes.

Each of these metrics summarize approximately 20,000 sampled molecules using one number, and so provide little information about the distribution of molecules produced. Therefore, following prior work (Polykovskiy et al., 2020; Seff et al., 2019), we also produce a KDE (kernel density estimate) plot for the distribution of certain properties of the sampled molecules. The properties we consider are the Quantitative Estimate of Drug-likeness score (QED, see § 4.3.2; Bickerton et al., 2012), the synthetic accessibility score (SA; Ertl

[1] For the evaluation in this chapter, we re-implemented the CVAE and GVAE models in PyTorch and found that our implementation was significantly better than (Kusner et al., 2017)'s published results. We believe this is down to being able to take advantage of some of the latest techniques for training these models (for example β-annealing – see e.g. Alemi et al., 2018; Higgins et al., 2017) as well as hyperparameter tuning. See Appendix D.4.2 for further details on the baselines used.

Table 5.1 *Table showing the percentage of valid molecules generated and then, conditioned on this, the uniqueness, novelty, and normalized quality (Brown et al., 2019, § 3.3) (all as %, higher better) as well as the FCD score (Fréchet ChemNet Distance, lower better; Preuer et al., 2018). Here, "normalized quality" means that we divide through by the fraction of molecules in the training set that pass the quality filters (see Section 4.3.1). For each model we generate the molecules by decoding from 20k prior samples from the latent space.*

Model Name	Validity (↑)	Uniqueness (↑)	Novelty (↑)	Quality (↑)	FCD (↓)
Training Data	100.0	100.0	0.0	100.0	0.21
SMILES LSTM (Segler et al., 2018a)	94.8	95.5	74.9	101.93	0.46
CVAE (Gómez-Bombarelli et al., 2018)	96.2	97.6	76.9	103.82	0.43
GVAE (Kusner et al., 2017)	74.4	97.8	82.7	98.98	0.89
GraphVAE (Simonovsky and Komodakis, 2018)	42.2	57.7	96.1	94.64	13.92
JT-VAE (Jin et al., 2018)	100.0	99.2	94.9	102.34	0.93
CGVAE (Liu et al., 2018)	100.0	97.8	97.9	45.64	14.26
MOLECULE CHEF (§ 4.2)	98.9	96.7	90.0	99.0	0.79
DoG-AE	100.0	98.3	92.9	95.5	0.83
DoG-Gen	100.0	97.7	88.4	101.6	0.45

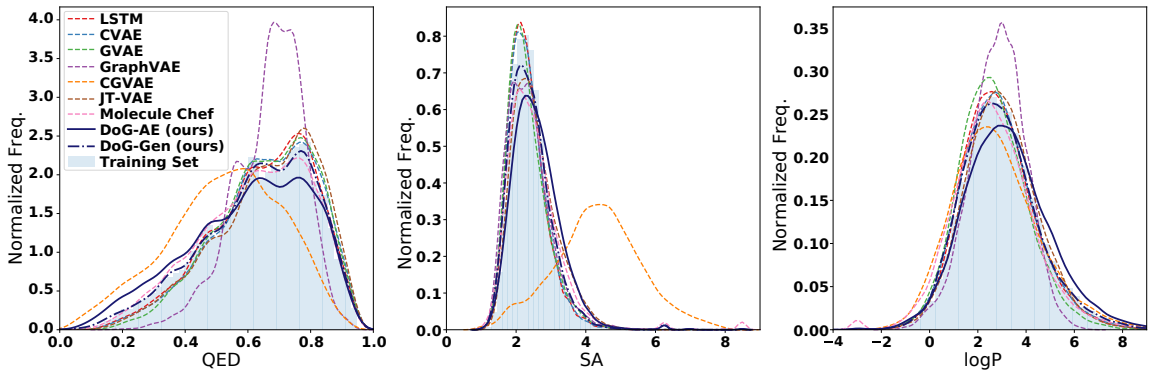


Figure 5.6 *KDE (kernel density estimate) plots for the distribution of the Quantitative Estimate of Drug-likeness score (QED; Bickerton et al., 2012), the synthetic accessibility score (SA; Ertl and Schuffenhauer, 2009), and the octanol-water partition coefficient (logP) for 20k molecules sampled from each of the various models. We also plot the distribution of these properties for the molecules in the training set (blue histogram). Plot done in same style as Seff et al. (2019, Fig. 3).*

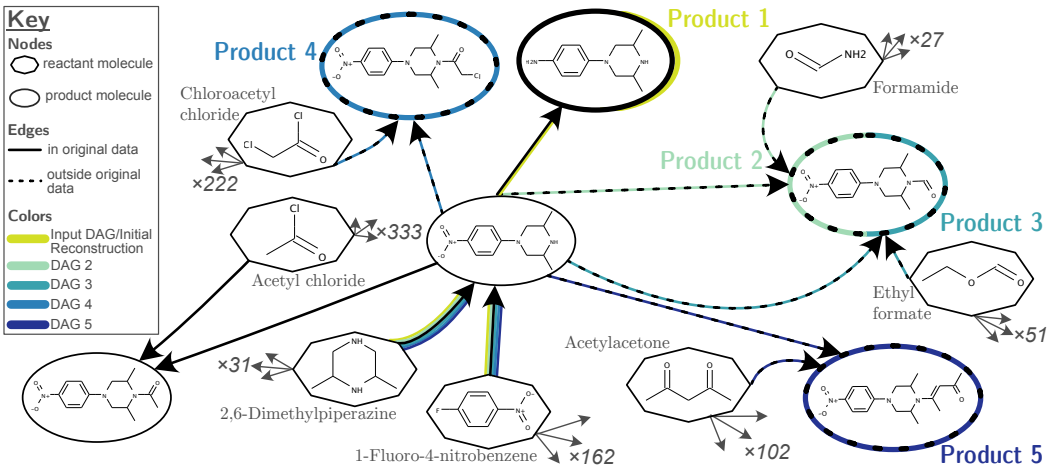


Figure 5.7 *Using a variant of the* DoG-AE *model, as we randomly walk in the latent space we decode out to similar DAGs nearby, unseen in training. Reactions and nodes that exist in our original dataset are outlined in solid lines, whereas those that have been discovered by our model are shown with dashed lines.*

and Schuffenhauer, 2009), and the octanol-water partition coefficient (logP; Wildman and Crippen, 1999). The results are shown in Figure 5.6. Apart from the GraphVAE and CGVAE, all models closely follow the distribution found in the training set, with again little to choose between them.

Qualitative evaluation of DoG-AE's latent space An important advantage of our models over the others is that they directly generate synthesis DAGs, indicating how a chosen molecule could be made. To visualize the properties of the latent space of DAGs learned by DoG-AE, we start from a training synthesis DAG and walk randomly in the latent space until we have output five different synthesis DAGs. We plot the combination of these DAGs, which can be seen as a reaction network, in Figure 5.7. We see that as we move around the latent space many of the synthesis DAGs have subgraphs that are isomorphic, resulting in similar final molecules.

5.4.2 Optimizing *synthesizable* molecules

We next look at how our model can be used for the global optimization of molecules with desirable properties. To evaluate our model, we compare its performance on a series of 10 optimization tasks from GuacaMol (Brown et al., 2019, § 3.2)[2] against the three best

[2]We also optimized again for the QED score favored by previous work (e.g. that of Jin et al., 2018; Kusner et al., 2017; You et al., 2018a). However, similar to Brown et al. (2019, § 7.5), we did not find this objective discriminative for objectively comparing the different approaches; DoG-Gen was able to obtain

reported models Brown et al. (2019, Table 2) found: (1) SMILES LSTM (Segler et al., 2018a), which does optimization via fine-tuning; (2) GraphGA (Jensen, 2019), a graph genetic algorithm (GA); and (3) SMILES GA (Yoshikawa et al., 2018), a SMILES-based GA. We train all methods on the same data, which is derived from USPTO and, as such, should give a strong bias for synthesizability.

We note that we should not expect our model to find the best molecule if judged solely on the GuacaMol task score: our model has to build up molecules from set building blocks and pre-learned reactions, which although reflecting the real-life process of molecule design, means that it is operating in a more constrained regime. However, the final property score is not the sole factor that is important when considering a proposed molecule. As we mentioned in the previous chapter, molecules need to be viable, meaning here that they need to (a) exist without degrading or reacting further (*i.e. be sufficiently stable*), and (b) be able to be created in practice (*i.e. be synthesizable*). To quantify (a) we consider using the quality filters proposed in Brown et al. (2019, § 3.3), also used last chapter. To quantify (b) we use computer-aided synthesis planning (Boda et al., 2007; Gao and Coley, 2020; Segler et al., 2018b). Specifically, we run a retrosynthesis tool on each molecule to see if a synthetic route can be found, and if so, how many steps are involved[3]. We also measure an aggregated synthesizability score over each step (see Appendix D.3), with a higher synthesizability score indicating that the individual reactions are closer to actual "observed" reactions and so hopefully more likely to work. All results are calculated on the top 100 molecules found by each method for each GuacaMol task.

The results of the experiments are shown in Figures 5.8 and 5.9, and Table 5.2 (see Appendix D.5 for further results). In Figure 5.8 we see that, disregarding synthesis, in general the Graph GA and SMILES LSTM approaches produce the best scoring molecules for the GuacaMol tasks. However, corroborating the findings of Gao and Coley (2020, Fig. S6), we note that the GA methods regularly suggest molecules for which no synthetic routes can be found. Our model, because it decodes synthesis DAGs, consistently finds high-scoring molecules while maintaining high synthesizability scores. Furthermore, Figure 5.9 shows that a high fraction of the molecules produced by our method pass the quality checks, whereas for some of the other methods the majority of molecules can fail.

the maximum commonly available QED score of 0.948 with a molecule represented by the SMILES string "`Cc1cc(F)ccc1NS(=O)(=O)c1ccc2c(c1)CCO2`".

[3]Note, like reaction predictors, these tools are imperfect, but still (we believe) offer a sensible current method for evaluating our models.

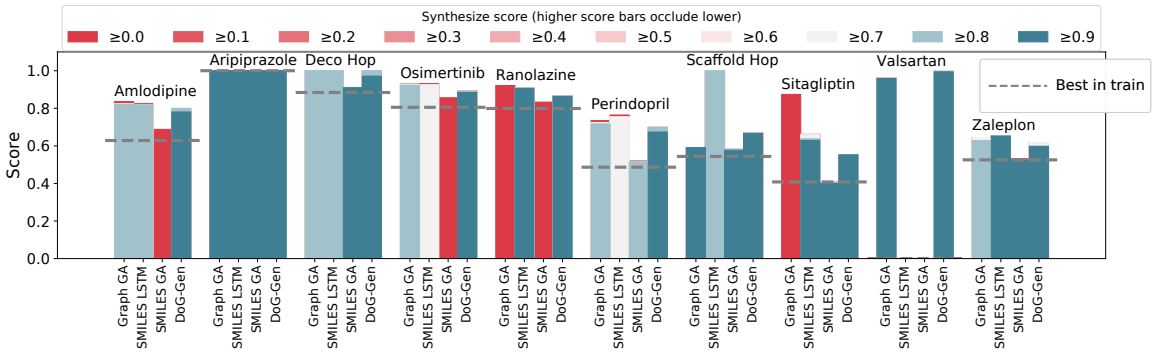


Figure 5.8 *The score of the best molecule found by the different approaches for 10 GuacaMol benchmark tasks (Brown et al., 2019, § 3.2) – task names are labeled above each set of bars. GuacaMol molecule scores (y-axis) range between 0 and 1, with 1 being the best. We also use colors to indicate the synthesizability score (see Appendix D.3) of the best molecule found. Note that bars representing a molecule within a higher synthesizability score bucket (e.g. blue) will occlude lower synthesizability score bars (e.g. red). The dotted gray lines represent the scores of the best molecule in our training set.*

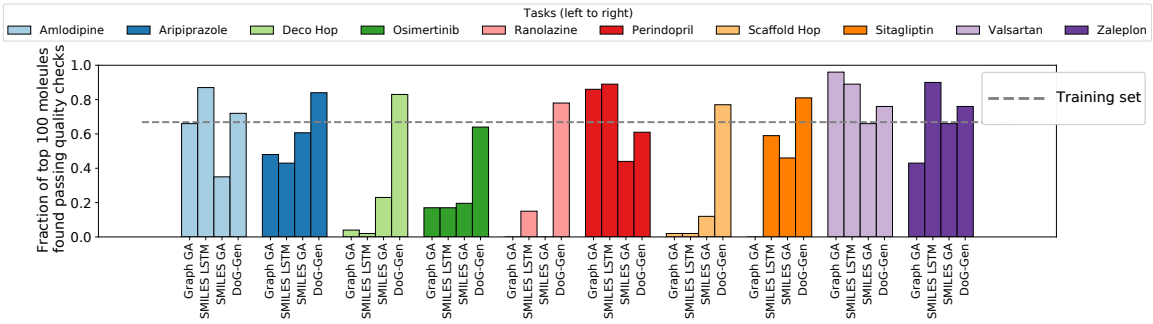


Figure 5.9 *The fraction of the top 100 molecules proposed that pass the quality filters, over a series of ten GuacaMol tasks (Brown et al., 2019, § 3.2). The fraction of molecules in our initial training set that pass the filters (which is 67%) is shown by the dotted gray line.*

Table 5.2 *Table showing metrics quantifying the stability and synthesizability of the molecules suggested by each method for the GuacaMol optimization tasks. The metrics we use include: the fraction of molecules for which a synthetic route is found, the mean synthesizability score, the median number of synthesis steps (for synthesizable molecules), and the fraction of molecules that pass the quality filters from Brown et al. (2019, § 3.3). The metrics are computed using the aggregation of the top 100 molecules for each GuacaMol task over all of the ten GuacaMol tasks we consider. For the baselines we use the implementations from GuacaMol (Brown et al., 2019). NB given that we are looking at "optimized" molecules, we no longer normalize the quality figures.*

	Frac. Synthesizable (↑)	Synth. Score (↑)	Median # Steps (↓)	Quality (↑)
Graph GA (Jensen, 2019)	0.42	0.33	6	0.36
SMILES LSTM (Segler et al., 2018a)	0.48	0.39	5	0.49
SMILES GA (Yoshikawa et al., 2018)	0.29	0.25	3	0.39
DoG-Gen	0.9	0.76	4	0.75

5.5 Discussion

In this section we discuss our approach. In particular, we discuss its relation to other recent work in ML and draw attention to some of its current limitations, describing how these motivate exciting future research directions to explore.

5.5.1 Related work

We have already discussed much of the related work for generating molecules in the background section of the previous chapter (§ 4.1). We will not repeat that discussion here but instead will describe some alternative ML-based approaches for generating synthesizable molecules (developed concurrently with the work described in this thesis) as well as briefly discuss how our work in this chapter relates to work in other application areas of ML.

Generating synthesizable molecules We are not alone in drawing attention to the importance of synthesizability in ML-based molecule design. Recently, there has been a series of other work incorporating reaction predictors with ML-based optimization techniques (Gottipati et al., 2020; Horwood and Noutahi, 2020; Korovina et al., 2020). Korovina et al. (2020) has a particular focus on sample efficiency, so we will discuss their approach in more detail in the section below. Gottipati et al. (2020); Horwood and Noutahi (2020) optimize molecules through reinforcement learning, with an action space consisting of reactions/reactants that can combine with a current molecule representing the state. However, both approaches are limited to linear synthesis trees, where intermediates can only react with a fixed set of building blocks rather than with other intermediates. We believe these approaches, which focus more on improving the underlying RL algorithms, are somewhat complimentary with the approach described here and could be combined in future work.

Other application areas Recently, in other application areas of ML, there has been much work for generating or editing graphs (including DAGs). A variety of different graph types have been considered, including citation networks and community graphs, as well as those representing neural network architectures or source code (Alvarez-Melis and Jaakkola, 2017; Chakraborty et al., 2020; Chen et al., 2018; Guo et al., 2018; You et al., 2018b; Zhang et al., 2019). DAGs have been generated in both a bottom-up (e.g. Zhang et al., 2019) and top-down (e.g. Chen et al., 2018) manner. Our approach is perhaps most related to Zhang et al. (2019), which develops an autoencoder model for DAGs representing neural network

architectures. However, synthesis DAGs have some key differences to those typically representing neural network architectures or source code, for instance nodes represent molecular graphs and should be unique within the DAG.

5.5.2 Sample efficiency

For optimization, one of the main limitations of our method, and this also holds true for the baselines we considered, is its poor sample efficiency. By poor sample efficiency, we mean that these methods require many queries to a property oracle when carrying out the optimization tasks. While properties such as the QED or those used in the GuacaMol tasks can be quickly computed by a computer, often interesting properties are far more expensive to evaluate. Ideally, one would therefore wish to evaluate them as infrequently as possible.

One possible approach to this problem might be to borrow ideas from Bayesian optimization (Shahriari et al., 2016), a principled optimization framework that accounts for sample efficiency. Bayesian optimization has been applied to molecule search in a variety of different ways (De Grave et al., 2008; Gómez-Bombarelli et al., 2018; Hernández-Lobato et al., 2017; Korovina et al., 2020; Williams et al., 2015). However, as far as we are aware, the only method that explicitly considers synthesizability, and generates novel molecules, is the work of Korovina et al. (2020). Korovina et al. (2020) performs a random walk on a reaction network, deciding which molecules to query next using Bayesian optimization. However, the random walk nature of their search means that this method only explores the reaction network locally. It would be interesting to see if combining their method with the ideas presented here allows one to explore this network more quickly , and so get the advantages of both approaches.

5.5.3 Improving the evaluation of molecule generation

Performing meaningful quantitative assessments of the different generative models of molecules is hard. We can break these difficulties down into (a) choosing suitable metrics, and (b) picking relevant baselines, both of which we shall now discuss.

Choosing suitable metrics In the last two chapters, to evaluate our approaches' abilities to generate new molecules we turned to the "soundness checks" used in previous works, i.e. assessing generated molecules in terms of validity, uniqueness, novelty, etc. (see Tables 4.1 & 5.1). Unfortunately, these checks have their limitations (Brown et al., 2019;Renz et al., 2019, § 2). This was perhaps most strikingly recently demonstrated by

Renz et al. (2019, Table 1), who showed that by trivially adding extra atoms to molecules in the training dataset, one could outperform many more complex models. This is not to say that we think that these checks should be abandoned: they still have their uses when one is mindful of their shortcomings. For instance, while it is possible for a bad model to pass these checks, it is unlikely that a good model would fail them, i.e. we could say that these checks have high specificity and low sensitivity for identifying bad models.

How can we come up with better metrics? One approach is to borrow ideas from how generative models are being evaluated in other application areas in machine learning. This has proved successful previously; for instance, the development of the FCD score (Preuer et al., 2018) built upon the Fréchet Inception Distance evaluation (Heusel et al., 2017; Salimans et al., 2016), used for assessing and comparing GANs (generative adversarial networks) producing natural images.

But perhaps a better approach for developing improved metrics is to start by thinking about what we actually want to use generative models of molecules for. If it is solely to discover new molecules, then it makes more sense to evaluate and compare the models on this task. This can be done using one of the objectives we considered in the optimization experiments, or even more recent benchmark-targets incorporating docking scores (Boitreaud et al., 2020; Cieplinski et al., 2020). A crucial challenge in this area is to design tasks that are significantly challenging enough to be interesting, yet not too computationally expensive to evaluate such that they inhibit fast experimentation.

Picking relevant baselines The other difficult task when assessing our models is choosing relevant baseline models to compare to. For instance, in this chapter we found that by re-implementing the CVAE and GVAE models, tuning the hyperparameters, and by using more modern autoencoder training techniques to enable the training of larger, autoregressive decoders, we were able to get much better performance than from the implementations of these models commonly used (§ 5.4.1). This is problematic, as when comparing generative models of molecules, we want to evaluate the differences attributable to the distinctions in the conceptual level in which they represent molecules, rather than variations arising due to hyperparameters. Unfortunately, this is often easier said than done: hyperparameter sweeps are expensive to perform and even here one would pick the best model based on an autoencoder loss, which may not correlate perfectly with other objective metrics.

Fortunately, progress in improving the situation has been made through the development of good-quality reference implementations and benchmark suites (Brown et al.,

2019; Polykovskiy et al., 2020). We therefore included baselines from these benchmarks here.

5.5.4 Multi-objective optimization

In evaluating our models in optimization tasks, we only considered optimizing for a single property at a time[4]. In practice, we may want to optimize for several objectives at once. For example, we might wish to find a molecule that is both effective and cheap to produce. This particular example would be fairly straightforward to do using our approach, given that we explicitly consider the construction sequence. For instance, we could easily assign costs to each building block, obtaining reasonable prices from a chemical vendor's catalog. We could then sum the costs for the building blocks used and add it to our original objective, optimizing this new, replacement objective in the same way.

However, in multi-objective optimization tasks one often does not want to find just a single optimum; instead, one may wish to obtain the "Pareto set", the set of points for which one cannot improve in one objective without doing worse in another (Collette and Siarry, 2004; Van Moffaert and Nowe, 2014). Therefore, an interesting avenue of research here would be to explore the use of other optimization algorithms, specifically designed for this task.

5.6 Summary

In this chapter we introduced a new model for generating molecules, DoG-Gen. DoG-Gen extended the ideas introduced in the last chapter with Molecule Chef, of generating molecules via virtual chemical reactions, to multiple steps of reactions. Specifically, it did this by generating a molecule synthesis DAG, describing how simple building block molecules recursively combine together to form more complex molecules of interest. We showed how we could use DoG-Gen as a component in larger ML architectures and frameworks. For instance, we showed how it could be used as the decoder in an autoencoder structure (DoG-AE), for sampling and interpolating in the space of reaction networks. In addition, when paired with a RL algorithm, it could be used for optimization, finding molecules that not only had good property scores, but also could be synthesized too.

[4]Implicitly these could depend on a set of several different, independent calculations but even so the relative weighting of these is fixed.

Chapter 6

Conclusions and Future Directions

This thesis has presented a new method for chemical reaction prediction and, using ML reaction predictors, brought center stage the idea of synthesis in ML-driven molecule generation. In this final chapter we summarize the contributions of our work and lay out possible future directions.

6.1 Summary of contributions

In this section we briefly summarize the contributions we have made in each chapter.

In **Chapter 3** we introduced a new model, ELECTRO, for *reaction prediction* on reactions exhibiting linear electron flow topology. Here, we demonstrated how we could design a ML model that predicted electron movements in an end-to-end manner. We then designed an approach to extract approximate reaction mechanisms from atom-mapped reaction SMILES strings, allowing us to train ELECTRO on large reaction datasets. Finally, we showed that ELECTRO compared favorably to the strongest baselines of the time (when predicting final products) and was able to learn basic chemical concepts, such as functional group selectivity, without explicit training.

In Chapters 4 and 5 we introduced new models for the *generative modeling of molecules*. **Chapter 4** introduced MOLECULE CHEF, a model that is able to address, simultaneously, two common problems in the search for new molecules: what molecule to make and how to make it. We showed that the sensible inductive bias of first selecting reactants before then transforming them under chemical reactions, allowed MOLECULE CHEF to generate a wide range of valid and stable molecules. Intriguingly, models such as MOLECULE CHEF also offer the opportunity to solve established problems such as retrosynthesis in new ways, and we presented an example of how we could MOLECULE CHEF to find alternatives to certain molecules with easier reactants to handle.

Chapter 5 extended the ideas introduced in MOLECULE CHEF to multiple reactions. We proposed a new model, DOG-GEN, to model complex synthesis plans. We explained how complex synthesis plans could be described using a synthesis DAG, and introduced an efficient serialization procedure over such a structure. Following this, we developed a hierarchical message passing routine over synthesis DAGs, and used this in conjunction with DOG-GEN to create an autoencoder, DOG-AE, useful for sampling and interpolating in the space of reaction networks. Finally, we showed that by combining DOG-GEN with RL-based optimization routines, we could find optimized molecules with comparable scores to those found by previous, unconstrained approaches, while ensuring that the molecules we suggested were synthesizable and stable too.

6.2 Future directions

We now turn to possible future directions of our work.

6.2.1 Levels of abstraction

A key theme in this thesis is that we can model molecules and their interactions at a variety of different levels of abstraction. These different abstractions can have different characteristics, provide different information, or be more natural to model. For instance, by modeling electron flows using ELECTRO in Chapter 4, we showed how we were able to predict the outcomes of new reactions in an end-to-end and interpretable manner, while ensuring that we always respected balanced atom counts. Likewise, in Chapter 5, we demonstrated that by generating molecules via virtual reaction schemes, more aligned with how a chemist would create a molecule in practice, we were able to generate molecules that performed competitively on optimization tasks while also being more synthesizable and stable too.

Abstractions elsewhere in chemistry An interesting avenue of research would be to take these ideas forward to other areas of chemistry or the natural sciences. An obvious idea might be molecule regression tasks, for instance predicting how well a molecule might "dock" with a protein. Chemistry is a mature field, such that many powerful, principled, and interpretable abstractions have been invented and refined over centuries of development. It is prudent to take advantage of these tools when designing new models. Often these ideas can also then feedback to other areas of ML. For instance, as discussed in Chapter 2, modern GNNs in ML hark back to many older ideas in chemoinformatics.

Combining representations Another possibility is to investigate how we can combine abstractions or representations. For instance, with reaction prediction, we have a whole hierarchy, or "stack", of levels with which we can represent the underlying process; models operating at different levels often trade off compute for accuracy. It would be interesting to develop models that can operate across the whole stack at once, dropping down to a lower level whenever their uncertainties suggest more accurate or precise calculations are required.

Sometimes we may wish to also use different representations or abstractions at the same time. This relates to multi-view or multimodal representation learning and dataset augmentation (Ngiam et al., 2011; Shorten and Khoshgoftaar, 2019; Wang et al., 2015). For instance, with the autoencoders used for molecules, it may be interesting to train multiple decoders simultaneously. In choosing a decoder, we are implicitly assigning a prior to what constitutes a "neighborhood" of a molecule, and so by using several and weighting their losses differently, we have another tool to structure the latent space.

A similar idea might also be useful in molecule regression tasks. If the task does not indicate a natural representation, then by training a separate model for each different representation and then marginalizing over them at test time, we might be able to obtain much more robust final representations and predictions.

6.2.2 Synthesis planning

Although in general we have argued that one should design molecules with synthesizability in mind, sometimes one might want to find a synthetic route for a molecule which one already knows has good properties, such as a natural product. Alternatively, it is sometimes beneficial, for instance due to environmental reasons, to look for new, alternative synthetic routes for products which one can already make. Currently, such problems might be tackled by top-down synthesis planners (Segler et al., 2018b; Szymkuć et al., 2016). These methods often work in a recursive manner. Starting from the final product node, they use tree search algorithms to gradually expand the current nodes in the tree until a complete route from the target molecule to building block molecules is found (or the method's compute budget runs out).

It would be interesting to explore extending some of the ideas proposed in Section 4.3.3 (where we used MOLECULE CHEF for retrosynthesis[1]) to learn a direct mapping from products to synthesis DAGs in a bottom-up manner (using ML parlance we might describe such a technique as "amortizing" the cost of retrosynthesis). Even if such an approach

[1] Some preliminary results of using DOG-GEN in a similar manner are also included in Appendix D.5.2

was unable to find a route to the target molecule, the intermediate products found might provide more useful building blocks for subsequent use in a top-down planner. Also, another related, possible future avenue to explore is treating retrosynthesis as an optimization task. We could use a similar fine-tuning technique to the one used in Section 5.3.1, initially pre-training a model for retrosynthesis but then adapting it for particular targets by fine-tuning.

6.2.3 Making ML methods more practical

A final theme of this thesis we wish to discuss is making generative models of molecules more practical. In particular, in Chapters 4 and 5, given the intrinsic link between the "design" and the "make" steps in molecule development, we brought synthesis instructions into the generative process itself. We showed how this meant that we were also more likely to generate stable molecules.

One enticing area of future research in this vein, is looking at how we can move beyond the computational proof of concepts provided here and experimentally validate these ideas in practice. Not only does this require collaboration with chemists but further ML development work too. We suggested some immediate considerations, solvable with ML, in the discussion of Section 5.5. However, other ML/computational work would probably be required too, such as designing methods to optimize reaction conditions. Most exciting of all is likely the design of solutions to the problems we do not even know exist yet, and in this manner such a research direction can drive intriguing work in many directions.

6.3 Conclusion

This thesis has argued that while working in the seams of chemistry and machine learning it is important to stitch together ideas from both fields. We demonstrated in the interlinked areas of reaction prediction and molecule design that building our ML models around abstractions from chemistry can aid their interpretability and extrapolation properties. While there are still many open challenges in these areas, we hope that the models that we have developed, and the ideas we have extolled, will act as a spring board in the development of future techniques.

References

Alemi, A., Poole, B., Fischer, I., Dillon, J., Saurous, R. A., and Murphy, K. (2018). Fixing a broken ELBO. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp.159–168. PMLR.

Alvarez-Melis, D. and Jaakkola, T. S. (2017). Tree-structured decoding with doubly-recurrent neural networks. In *International Conference on Learning Representations 2017*.

Anderson, B., Hy, T. S., and Kondor, R. (2019). Cormorant: covariant molecular neural networks. In *Advances in Neural Information Processing Systems 32*, pp.14537–14546. Curran Associates, Inc.

Ash, S., Cline, M. A., Homer, R. W., Hurst, T., and Smith, G. B. (1997). SYBYL line notation (SLN): a versatile language for chemical structure representation. *Journal of Chemical Information and Computer Sciences*, 37(1):71–79.

Awale, M. and Reymond, J.-L. (2014). Atom pair 2D-fingerprints perceive 3D-molecular shape and pharmacophores for very fast virtual screening of ZINC and GDB-17. *Journal of Chemical Information and Modeling*, 54(7):1892–1907.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv:1409.0473.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks. arXiv:1806.01261.

Bauer, J., Fontain, E., Forstmeyer, D., and Ugi, I. (1988). Interactive generation of organic reactions by IGOR 2 and the PC-assisted discovery of a new reaction. *Tetrahedron Computer Methodology*, 1(2):129–132.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems 28*, pp.1171–1179. Curran Associates, Inc.

Bergeler, M., Simm, G. N., Proppe, J., and Reiher, M. (2015). Heuristics-guided exploration of reaction mechanisms. *Journal of Chemical Theory and Computation*, 11(12):5712–5722.

Bi, H., Wang, H., Shi, C., and Tang, J. (2020). Non-autoregressive electron flow generation for reaction prediction. arXiv:2012.12124.

Bickerton, G. R., Paolini, G. V., Besnard, J., Muresan, S., and Hopkins, A. L. (2012). Quantifying the chemical beauty of drugs. *Nature Chemistry*, 4(2):90–98.

Bjerrum, E. J. (2017). SMILES enumeration as data augmentation for neural network modeling of molecules. arXiv:1703.07076.

Bloem-Reddy, B. and Teh, Y. W. (2020). Probabilistic symmetry and invariant neural networks. *Journal of Machine Learning Research*, 21(90):1–61.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426.

Boda, K., Seidel, T., and Gasteiger, J. (2007). Structure and reaction based evaluation of synthetic accessibility. *Journal of Computer-Aided Molecular Design*, 21(6):311–325.

Boitreaud, J., Mallet, V., Oliver, C., and Waldispühl, J. (2020). OptiMol: optimization of binding affinities in chemical space for drug discovery. *Journal of Chemical Information and Modeling*, 60(12):5658–5666.

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. (2016). Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pp.10–21. Association for Computational Linguistics.

Bradshaw, J., Kusner, M. J., Paige, B., Segler, M. H. S., and Hernández-Lobato, J. M. (2019a). A generative model for electron paths. In *International Conference on Learning Representations 2019*.

Bradshaw, J., Paige, B., Kusner, M., Segler, M., and Hernández-Lobato, J. M. (2020). Barking up the right tree: an approach to search over molecule synthesis DAGs. In *Advances in Neural Information Processing Systems 33 (To Appear)*.

Bradshaw, J., Paige, B., Kusner, M. J., Segler, M. H. S., and Hernández-Lobato, J. M. (2019b). A model to search for synthesizable molecules. In *Advances in Neural Information Processing Systems 32*, pp.7937–7949. Curran Associates, Inc.

Brockschmidt, M. (2020). GNN-FiLM: graph neural networks with feature-wise linear modulation. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pp.1144–1152. PMLR.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42.

Brown, N. (2009). Chemoinformatics—an introduction for computer scientists. *ACM Computing Surveys*, 41(2):8.

Brown, N., Fiscato, M., Segler, M. H. S., and Vaucher, A. C. (2019). GuacaMol: benchmarking models for de novo molecular design. *Journal of Chemical Information and Modeling*, 59(3):1096–1108.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. arXiv:1312.6203.

Bunnage, M. E. (2011). Getting pharmaceutical R&D back on target. *Nature Chemical Biology*, 7(6):335–339.

Button, A., Merk, D., Hiss, J. A., and Schneider, G. (2019). Automated de novo molecular design by hybrid machine intelligence and rule-driven chemical synthesis. *Nature Machine Intelligence*, 1(7):307–315.

Chakraborty, S., Allamanis, M., and Ray, B. (2020). CODIT: code editing with tree-based neural machine translation. *IEEE Transactions on Software Engineering*.

Chen, B., Li, C., Dai, H., and Song, L. (2020). Retro*: learning retrosynthetic planning with neural guided A* search. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pp.1608–1616. PMLR.

Chen, B., Shen, T., Jaakkola, T. S., and Barzilay, R. (2019). Learning to make generalizable and diverse predictions for retrosynthesis. arXiv:1910.09688.

Chen, J. H. and Baldi, P. (2009). No electron left behind: a rule-based expert system to predict chemical reactions and reaction mechanisms. *Journal of Chemical Information and Modeling*, 49(9):2034–2043.

Chen, X., Liu, C., and Song, D. (2018). Tree-to-tree neural networks for program translation. In *Advances in Neural Information Processing Systems 31*, pp.2547–2557. Curran Associates, Inc.

Chevillard, F. and Kolb, P. (2015). SCUBIDOO: a large yet screenable and easily searchable database of computationally created chemical compounds optimized toward high likelihood of synthetic tractability. *Journal of Chemical Information and Modeling*, 55(9):1824–1835.

Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014a). On the properties of neural machine translation: encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pp. 103–111. Association for Computational Linguistics.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). 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.

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

Clayden, J., Greeves, N., and Warren, S. (2012). *Organic Chemistry*. Oxford University Press, 2nd edition.

Coelho, L. P. (2017). Jug: software for parallel reproducible computation in Python. *Journal of Open Research Software*, 5.

Coley, C. W., Barzilay, R., Jaakkola, T. S., Green, W. H., and Jensen, K. F. (2017a). Prediction of organic reaction outcomes using machine learning. *ACS Central Science*, 3(5):434–443.

Coley, C. W., Jin, W., Rogers, L., Jamison, T. F., Jaakkola, T. S., Green, W. H., Barzilay, R., and Jensen, K. F. (2019). A graph-convolutional neural network model for the prediction of chemical reactivity. *Chemical Science*, 10(2):370–377.

Coley, C. W., Rogers, L., Green, W. H., and Jensen, K. F. (2017b). Computer-assisted retrosynthesis based on molecular similarity. *ACS Central Science*, 3(12):1237–1245.

Collette, Y. and Siarry, P. (2004). *Multiobjective Optimization: Principles and Case Studies*. Springer-Verlag Berlin Heidelberg.

Cook, A., Johnson, A. P., Law, J., Mirzazadeh, M., Ravitz, O., and Simon, A. (2012). Computer-aided synthesis design: 40 years on. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 2(1):79–107.

Corey, E. J. (1971). Centenary lecture. Computer-assisted analysis of complex synthetic problems. *Quarterly Reviews, Chemical Society*, 25(4):455–482.

Corey, E. J. and Wipke, W. T. (1969). Computer-assisted design of complex organic syntheses. *Science*, 166(3902):178–192.

Corey, E. J., Wipke, W. T., Cramer, R. D., and Howe, W. J. (1972). Techniques for perception by a computer of synthetically significant structural features in complex molecules. *Journal of the American Chemical Society*, 94(2):431–439.

Cramer, C. J. (2013). *Essentials of Computational Chemistry: Theories and Models*. John Wiley & Sons.

Dahl, G. E., Jaitly, N., and Salakhutdinov, R. (2014). Multi-task neural networks for QSAR predictions. arXiv:1406.1231.

Dai, H., Dai, B., and Song, L. (2016). Discriminative embeddings of latent variable models for structured data. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pp.2702–2711. PMLR.

Dai, H., Li, C., Coley, C., Dai, B., and Song, L. (2019). Retrosynthesis prediction with Conditional Graph Logic Network. In *Advances in Neural Information Processing Systems 32*, pp.8872–8882. Curran Associates, Inc.

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

Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995). The Helmholtz machine. *Neural Computation*, 7(5):889–904.

de Boer, P.-T., Kroese, D. P., Mannor, S., and Rubinstein, R. Y. (2005). A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67.

De Cao, N. and Kipf, T. (2018). MolGAN: an implicit generative model for small molecular graphs. In *Theoretical Foundations and Applications of Deep Generative Models - ICML 2018 Workshop*.

De Grave, K., Ramon, J., and De Raedt, L. (2008). Active learning for high throughput screening. In *Discovery Science*, pp.185–196. Springer-Verlag Berlin Heidelberg.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems 29*, pp.3844–3852. Curran Associates Inc.

DeWitte, R. S. and Shakhnovich, E. I. (1996). SMoG: de novo design method based on simple, fast, and accurate free energy estimates. 1. Methodology and supporting evidence. *Journal of the American Chemical Society*, 118(47):11733–11744.

DiMasi, J. A., Grabowski, H. G., and Hansen, R. W. (2016). Innovation in the pharmaceutical industry: new estimates of R&D costs. *Journal of Health Economics*, 47:20–33.

Do, K., Tran, T., and Venkatesh, S. (2019). Graph transformation policy network for chemical reaction prediction. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.750–760.

Durant, J. L., Leland, B. A., Henry, D. R., and Nourse, J. G. (2002). Reoptimization of MDL keys for use in drug discovery. *Journal of Chemical Information and Computer Sciences*, 42(6):1273–1280.

Duvenaud, D., Maclaurin, D., Aguilera-Iparraguirre, J., Gómez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems 28*, pp.2224–2232. Curran Associates, Inc.

Effland, R. C., Gardner, B. A., and Strupczewski, J. (1981). Synthesis of 2,3-dihydrospiro[benzofuran-2,4' -piperidines] and 2,3-dihydrospiro[benzofuran-2,3'-pyrrolidines]. *Journal of Heterocyclic Chemistry*, 18(4):811–814.

Ellis, F. (2002). *Paracetamol - a curriculum resource*. Royal Society of Chemistry.

Engel, T. and Gasteiger, J. (2018). *Chemoinformatics: Basic Concepts and Methods*. John Wiley & Sons.

Errica, F., Podda, M., Bacciu, D., and Micheli, A. (2020). A fair comparison of graph neural networks for graph classification. In *International Conference on Learning Representations 2020*.

Ertl, P. and Schuffenhauer, A. (2009). Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of Cheminformatics*, 1(1):8.

Favre, H. A. and Powell, W. H. (2014). *Nomenclature of Organic Chemistry*. The Royal Society of Chemistry.

Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *Representation Learning on Graphs and Manifolds ICLR 2019 Workshop*.

Fontain, E. and Reitsam, K. (1991). The generation of reaction networks with RAIN. 1. The reaction generator. *Journal of Chemical Information and Computer Sciences*, 31(1):96–101.

Fooshee, D., Mood, A., Gutman, E., Tavakoli, M., Urban, G., Liu, F., Huynh, N., Van Vranken, D., and Baldi, P. (2018). Deep learning for chemical reaction prediction. *Molecular Systems Design & Engineering*, 3(3):442–452.

Gao, H., Struble, T. J., Coley, C. W., Wang, Y., Green, W. H., and Jensen, K. F. (2018). Using machine learning to predict suitable conditions for organic reactions. *ACS Central Science*, 4(11):1465–1476.

Gao, K., Nguyen, D. D., Sresht, V., Mathiowetz, A. M., Tu, M., and Wei, G.-W. (2020). Are 2D fingerprints still valuable for drug discovery? *Physical Chemistry Chemical Physics*, 22(16):8373–8390.

Gao, W. and Coley, C. W. (2020). The synthesizability of molecules proposed by generative models. *Journal of Chemical Information and Modeling*, 60(12):5714–5723.

Gasteiger, J. (2003). *Handbook of Chemoinformatics: From Data to Knowledge in 4 Volumes*. WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim.

Gasteiger, J. and Engel, T. (2003). *Chemoinformatics: A Textbook*. WILEY-VCH Verlag GmbH & Co. KGaA.

Gasteiger, J., Hutchings, M. G., Christoph, B., Gann, L., Hiller, C., Löw, P., Marsili, M., Saller, H., and Yuki, K. (1987). A new treatment of chemical reactivity: development of EROS, an expert system for reaction prediction and synthesis design. In *Organic Synthesis, Reactions and Mechanisms*, pp.19–73. Springer-Verlag Berlin Heidelberg.

Gershman, S. J. and Goodman, N. D. (2014). Amortized inference in probabilistic reasoning. In *Proceedings of the 36th Annual Conference of the Cognitive Science Society*.

Ghahramani, Z. (2004). Unsupervised learning. In *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*, pp.72–112. Springer-Verlag Berlin Heidelberg.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp.1263–1272. PMLR.

Goh, G. B., Hodas, N., Siegel, C., and Vishnu, A. (2018). SMILES2vec: predicting chemical properties from text representations. arXiv:1712.02034.

Goh, G. B., Siegel, C., Vishnu, A., Hodas, N. O., and Baker, N. (2017). Chemception: a deep neural network with minimal chemistry knowledge matches the performance of expert-developed QSAR/QSPR models. arXiv:1706.06689.

Gómez-Bombarelli, R., Aguilera-Iparraguirre, J., Hirzel, T. D., Duvenaud, D., Maclaurin, D., Blood-Forsythe, M. A., Chae, H. S., Einzinger, M., Ha, D.-G., Wu, T., Markopoulos, G., Jeon, S., Kang, H., Miyazaki, H., Numata, M., Kim, S., Huang, W., Hong, S. I., Baldo, M., Adams, R. P., and Aspuru-Guzik, A. (2016). Design of efficient molecular organic light-emitting diodes by a high-throughput virtual screening and experimental approach. *Nature Materials*, 15(10):1120–1127.

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., and Aspuru-Guzik, A. (2018). Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. The MIT Press.

Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings of 2005 IEEE International Joint Conference on Neural Networks*, volume 2, pp.729–734.

Gottipati, S. K., Sattarov, B., Niu, S., Pathak, Y., Wei, H., Liu, S., Thomas, K. M. J., Blackburn, S., Coley, C. W., Tang, J., Chandar, S., and Bengio, Y. (2020). Learning to navigate the synthetically accessible chemical space using reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pp.3668–3679. PMLR.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Scholkopf, B., and Smola, A. (2012). A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773.

Grzybowski, B. A., Bishop, K. J. M., Kowalczyk, B., and Wilmer, C. E. (2009). The 'wired' universe of organic chemistry. *Nature Chemistry*, 1(1):31–36.

Guimaraes, G. L., Sanchez-Lengeling, B., Outeiral, C., Farias, P. L. C., and Aspuru-Guzik, A. (2017). Objective-reinforced generative adversarial networks (ORGAN) for sequence generation models. arXiv:1705.10843.

Guo, X., Wu, L., and Zhao, L. (2018). Deep graph translation. arXiv:1805.09980.

Hähnke, V. D., Kim, S., and Bolton, E. E. (2018). PubChem chemical structure standardization. *Journal of Cheminformatics*, 10(1):36.

Hamilton, W. L. (2020). Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30*, pp.1024–1034. Curran Associates, Inc.

Hartenfeller, M. and Schneider, G. (2011). Enabling future drug discovery by de novo design. *WIREs Computational Molecular Science*, 1(5):742–759.

Hartenfeller, M., Zettl, H., Walter, M., Rupp, M., Reisen, F., Proschak, E., Weggen, S., Stark, H., and Schneider, G. (2012). DOGS: reaction-driven de novo design of bioactive compounds. *PLOS Computational Biology*, 8(2):e1002380.

Hawkins, P. C. D. (2017). Conformation generation: the state of the art. *Journal of Chemical Information and Modeling*, 57(8):1747–1756.

Heller, S., McNaught, A., Stein, S., Tchekhovskoi, D., and Pletnev, I. (2013). InChI - the worldwide chemical structure identifier standard. *Journal of Cheminformatics*, 5(1):7.

Heller, S. R., McNaught, A., Pletnev, I., Stein, S., and Tchekhovskoi, D. (2015). InChI, the IUPAC International Chemical Identifier. *Journal of Cheminformatics*, 7:23.

Herges, R. (1994a). Coarctate transition states: the discovery of a reaction principle. *Journal of Chemical Information and Computer Sciences*, 34(1):91–102.

Herges, R. (1994b). Organizing principle of complex reactions and theory of coarctate transition states. *Angewandte Chemie*, 33(3):255–276.

Herges, R. (2015). Coarctate and pseudocoarctate reactions: stereochemical rules. *The Journal of Organic Chemistry*, 80(23):11869–11876.

Hernández-Lobato, J. M., Requeima, J., Pyzer-Knapp, E. O., and Aspuru-Guzik, A. (2017). Parallel and distributed Thompson sampling for large-scale accelerated exploration of chemical space. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp.1470–1479. PMLR.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *Advances in Neural Information Processing Systems 30*, pp.6626–6637. Curran Associates, Inc.

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

Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, minimum description length and Helmholtz free energy. In *Advances in Neural Information Processing Systems 6*, pp.3–10. Morgan-Kaufmann.

Hiss, J. A., Reutlinger, M., Koch, C. P., Perna, A. M., Schneider, P., Rodrigues, T., Haller, S., Folkers, G., Weber, L., Baleeiro, R. B., Walden, P., Wrede, P., and Schneider, G. (2014). Combinatorial chemistry by ant colony optimization. *Future Medicinal Chemistry*, 6(3):267–280.

Holenz, J., editor (2016). *Lead Generation*. Methods and Principles in Medicinal Chemistry. Wiley-VCH Verlag GmbH & Co. KGaA.

Horwood, J. and Noutahi, E. (2020). Molecular design in synthetically accessible chemical space via deep reinforcement learning. *ACS Omega*, 5(51):32984–32994.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. (2020a). Open Graph Benchmark: datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems 33 (To Appear)*.

Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., and Leskovec, J. (2020b). Strategies for pre-training graph neural networks. In *International Conference on Learning Representations 2020*.

Humbeck, L., Weigang, S., Schäfer, T., Mutzel, P., and Koch, O. (2018). CHIPMUNK: a virtual synthesizable small-molecule library for medicinal chemistry, exploitable for protein-protein interaction modulators. *ChemMedChem*, 13(6):532–539.

Hunt, P. (2007). *Neglected diseases: a human rights analysis*. World Health Organization.

Ingraham, J., Garg, V., Barzilay, R., and Jaakkola, T. (2019). Generative models for graph-based protein design. In *Advances in Neural Information Processing Systems 32*, pp. 15820–15831. Curran Associates, Inc.

Ioset, J.-R. and Chang, S. (2011). Drugs for neglected diseases initiative model of drug development for neglected diseases: current status and future challenges. *Future Medicinal Chemistry*, 3(11):1361–1371.

Irwin, J. J., Sterling, T., Mysinger, M. M., Bolstad, E. S., and Coleman, R. G. (2012). ZINC: a free tool to discover chemistry for biology. *Journal of Chemical Information and Modeling*, 52(7):1757–1768.

IUPAC (2014). *IUPAC Compendium of Chemical Terminology (Gold Book)*. https://goldbook.iupac.org/, online edition.

Jacob, P.-M. and Lapkin, A. (2018). Statistics of the network of organic chemistry. *Reaction Chemistry & Engineering*, 3(1):102–118.

Jansen, M. and Schön, J. C. (2006). "Design" in chemical synthesis—an illusion? *Angewandte Chemie, International Edition*, 45(21):3406–3412.

Janz, D., van der Westhuizen, J., Paige, B., Kusner, M. J., and Hernández-Lobato, J. M. (2018). Learning a generative model for validity in complex discrete structures. In *International Conference on Learning Representations 2018*.

Jastrzębski, S., Leśniak, D., and Czarnecki, W. M. (2016). Learning to SMILE(S). In *International Conference on Learning Representations 2016 - Workshop track*.

Jensen, F. (2017). *Introduction to Computational Chemistry*. John Wiley & Sons.

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

Jin, W., Barzilay, R., and Jaakkola, T. (2018). Junction tree variational autoencoder for molecular graph generation. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp.2323–2332. PMLR.

Jin, W., Barzilay, R., and Jaakkola, T. (2020a). Hierarchical generation of molecular graphs using structural motifs. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pp.4839–4848. PMLR.

Jin, W., Barzilay, R., and Jaakkola, T. (2020b). Multi-objective molecule generation using interpretable substructures. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pp.4849–4859. PMLR.

Jin, W., Coley, C. W., Barzilay, R., and Jaakkola, T. (2017). Predicting organic reaction outcomes with Weisfeiler-Lehman Network. In *Advances in Neural Information Processing Systems 30*, pp.2607–2616. Curran Associates, Inc.

Jin, W., Yang, K., Barzilay, R., and Jaakkola, T. (2019). Learning multimodal graph-to-graph translation for molecular optimization. In *International Conference on Learning Representation 2019*.

Johnson, D. D. (2017). Learning graphical state transitions. In *International Conference on Learning Representations 2017*.

Kadurin, A., Aliper, A., Kazennov, A., Mamoshina, P., Vanhaelen, Q., Khrabrov, K., and Zhavoronkov, A. (2017). The cornucopia of meaningful leads: applying deep adversarial autoencoders for new molecule development in oncology. *Oncotarget*, 8(7):10883–10890.

Kajino, H. (2019). Molecular hypergraph grammar with its application to molecular optimization. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pp.3183–3191. PMLR.

Kayala, M. A., Azencott, C.-A., Chen, J. H., and Baldi, P. (2011). Learning to predict chemical reactions. *Journal of Chemical Information and Modeling*, 51(9):2209–2222.

Kayala, M. A. and Baldi, P. (2012). ReactionPredictor: prediction of complex chemical reactions at the mechanistic level using machine learning. *Journal of Chemical Information and Modeling*, 52(10):2526–2540.

Kayala, M. A. and Baldi, P. F. (2011). A machine learning approach to predict chemical reactions. In *Advances in Neural Information Processing Systems 24*, pp.747–755. Curran Associates, Inc.

Kearnes, S., McCloskey, K., Berndl, M., Pande, V., and Riley, P. (2016). Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-Aided Molecular Design*, 30(8):595–608.

Kermack, W. O. and Robinson, R. (1922). LI.—An explanation of the property of induced polarity of atoms and an interpretation of the theory of partial valencies on an electronic basis. *Journal of the Chemical Society, Transactions*, 121(0):427–440.

Kim, Y., Kim, J. W., Kim, Z., and Kim, W. Y. (2018). Efficient prediction of reaction paths through molecular graph and reaction network analysis. *Chemical Science*, 9(4):825–835.

Kingma, D. P. and Ba, J. (2014). Adam: a method for stochastic optimization. arXiv:1412.6980.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. arXiv:1312.6114v10.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations 2017*.

Kireev, D. B. (1995). ChemNet: a novel neural network based method for graph/property mapping. *Journal of Chemical Information and Computer Sciences*, 35(2):175–180.

Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A. (2017). OpenNMT: open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pp.67–72. Association for Computational Linguistics.

Kola, I. and Landis, J. (2004). Can the pharmaceutical industry reduce attrition rates? *Nature Reviews Drug Discovery*, 3(8):711–715.

Korovina, K., Xu, S., Kandasamy, K., Neiswanger, W., Poczos, B., Schneider, J., and Xing, E. P. (2020). ChemBO: Bayesian optimization of small organic molecules with synthesizable recommendations. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108, pp.3393–3403. PMLR.

Krenn, M., Häse, F., Nigam, A., Friederich, P., and Aspuru-Guzik, A. (2020). Self-referencing embedded strings (SELFIES): a 100% robust molecular string representation. *Machine Learning: Science and Technology*, 1(4):045024.

Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. (2017). Grammar variational autoencoder. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp.1945–1954. PMLR.

Law, J., Zsoldos, Z., Simon, A., Reid, D., Liu, Y., Khew, S. Y., Johnson, A. P., Major, S., Wade, R. A., and Ando, H. Y. (2009). Route designer: a retrosynthetic analysis tool utilizing automated retrosynthetic rule generation. *Journal of Chemical Information and Modeling*, 49(3):593–602.

Leach, A. R. and Gillet, V. J. (2007). *An Introduction to Chemoinformatics*. Springer.

Lee, A. A., Yang, Q., Sresht, V., Bolgar, P., Hou, X., Klug-McLeod, J. L., and Butler, C. R. (2019a). Molecular Transformer unifies reaction prediction and retrosynthesis across pharma chemical space. *Chemical Communications*, 55(81):12152–12155.

Lee, J., Lee, I., and Kang, J. (2019b). Self-attention graph pooling. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pp.3734–3743. PMLR.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2015). Gated Graph Sequence Neural Networks. arXiv:1511.05493.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. (2018a). Learning deep generative models of graphs. arXiv:1803.03324.

Li, Y., Zhang, L., and Liu, Z. (2018b). Multi-objective de novo drug design with conditional graph generative model. *Journal of Cheminformatics*, 10(1):33.

Liao, R., Li, Y., Song, Y., Wang, S., Nash, C., Hamilton, W. L., Duvenaud, D., Urtasun, R., and Zemel, R. S. (2019). Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems 32*, pp. 4255–4265. Curran Associates, Inc.

Lichtenberg, F. R. (2005). The impact of new drug launches on longevity: evidence from longitudinal, disease-level data from 52 countries, 1982-2001. *International Journal of Health Care Finance and Economics*, 5(1):47–73.

Liu, Q., Allamanis, M., Brockschmidt, M., and Gaunt, A. L. (2018). Constrained graph variational autoencoders for molecule design. In *Advances in Neural Information Processing Systems 31*, pp.7795–7804. Curran Associates, Inc.

Lowe, D. M. (2012). *Extraction of chemical structures and reactions from the literature*. PhD thesis, University of Cambridge.

Luo, Z., Wang, R., and Lai, L. (1996). RASSE: a new method for structure-based drug design. *Journal of Chemical Information and Computer Sciences*, 36(6):1187–1194.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp.1412–1421. Association for Computational Linguistics.

Ma, J., Sheridan, R. P., Liaw, A., Dahl, G. E., and Svetnik, V. (2015). Deep neural nets as a method for quantitative structure-activity relationships. *Journal of Chemical Information and Modeling*, 55(2):263–274.

Madhawa, K., Ishiguro, K., Nakago, K., and Abe, M. (2019). GraphNVP: an invertible flow model for generating molecular graphs. arXiv:1905.11600.

Mansimov, E., Mahmood, O., Kang, S., and Cho, K. (2019). Molecular geometry prediction using a deep generative graph neural network. *Scientific Reports*, 9(1):20381.

Mayr, A., Klambauer, G., Unterthiner, T., Steijaert, M., Wegner, J. K., Ceulemans, H., Clevert, D.-A., and Hochreiter, S. (2018). Large-scale comparison of machine learning methods for drug target prediction on ChEMBL. *Chemical Science*, 9(24):5441–5451.

Maziarka, Ł., Danel, T., Mucha, S., Rataj, K., Tabor, J., and Jastrzębski, S. (2020). Molecule Attention Transformer. arXiv:2002.08264.

McKay, B. D. (1981). Practical graph isomorphism. *Congressus Numerantium*, 30:45–87.

Mercado, R., Rastemo, T., Lindelöf, E., Klambauer, G., Engkvist, O., Chen, H., and Bjerrum, E. J. (2020). Graph networks for molecular design. *Machine Learning: Science and Technology*, 2(2):025023.

Merk, D., Friedrich, L., Grisoni, F., and Schneider, G. (2018a). De novo design of bioactive small molecules by artificial intelligence. *Molecular Informatics*, 37(1-2):1700153.

Merk, D., Grisoni, F., Friedrich, L., and Schneider, G. (2018b). Tuning artificial intelligence on the de novo design of natural-product-inspired retinoid X receptor modulators. *Communications Chemistry*, 1(1):68.

Merkwirth, C. and Lengauer, T. (2005). Automatic generation of complementary descriptors with molecular graph networks. *Journal of Chemical Information and Modeling*, 45(5):1159–1168.

Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., and Bronstein, M. M. (2017). Geometric deep learning on graphs and manifolds using mixture model CNNs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.5425–5434. IEEE.

Morgan, H. L. (1965). The generation of a unique machine description for chemical structures-a technique developed at Chemical Abstracts Service. *Journal of Chemical Documentation*, 5(2):107–113.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. (2019). Weisfeiler and Leman go neural: higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33(01), pp.4602–4609. AAAI Press.

Muratov, E. N., Bajorath, J., Sheridan, R. P., Tetko, I. V., Filimonov, D., Poroikov, V., Oprea, T. I., Baskin, I. I., Varnek, A., Roitberg, A., Isayev, O., Curtarolo, S., Fourches, D., Cohen, Y., Aspuru-Guzik, A., Winkler, D. A., Agrafiotis, D., Cherkasov, A., and Tropsha, A. (2020). QSAR without borders. *Chemical Society Reviews*, 49(11):3716.

Musa, O. M., editor (2016). *Handbook of Maleic Anhydride Based Materials: Syntheses, Properties and Applications*. Springer International Publishing.

Nam, J. and Kim, J. (2016). Linking the neural machine translation and the prediction of organic chemistry reactions. arXiv:1612.09529.

Nandi, S., McAnanama-Brereton, S. R., Waller, M. P., and Anoop, A. (2017). A tabu-search based strategy for modeling molecular aggregates and binary reactions. *Computational and Theoretical Chemistry*, 1111:69–81.

Neil, D., Segler, M., Guasch, L., Ahmed, M., Plumbley, D., Sellwood, M., and Brown, N. (2018). Exploring deep recurrent models with reinforcement learning for molecule design. In *International Conference on Learning Representations - Workshop Track*.

Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H., and Ng, A. Y. (2011). Multimodal deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pp.689–696. Omnipress.

Nishibata, Y. and Itai, A. (1991). Automatic creation of drug candidate structures based on receptor structure. Starting point for artificial lead generation. *Tetrahedron*, 47(43):8985–8990.

Niu, S. and Hall, M. B. (2000). Theoretical studies on reactions of transition-metal complexes. *Chemical Reviews*, 100(2):353–406.

O'Boyle, N. and Dalke, A. (2018). DeepSMILES: an adaptation of SMILES for use in machine-learning of chemical structures.

O'Boyle, N. M. (2012). Towards a universal SMILES representation - a standard method to generate canonical SMILES based on the InChI. *Journal of Cheminformatics*, 4(1):22.

O'Hagan, D. and Lloyd, D. (2010). The iconic curly arrow. *Chemistry World*, April:54–57.

Olivecrona, M., Blaschke, T., Engkvist, O., and Chen, H. (2017). Molecular de-novo design through deep reinforcement learning. *Journal of Cheminformatics*, 9(1):48.

Pammolli, F., Magazzini, L., and Riccaboni, M. (2011). The productivity crisis in pharmaceutical R&D. *Nature Reviews Drug Discovery*, 10(6):428–438.

Pattanaik, L., Ganea, O.-E., Coley, I., Jensen, K. F., Green, W. H., and Coley, C. W. (2020a). Message passing networks for molecules with tetrahedral chirality. In *Machine Learning for Molecules Workshop at NeurIPS 2020*.

Pattanaik, L., Ingraham, J. B., Grambow, C. A., and Green, W. H. (2020b). Generating transition states of isomerization reactions with deep learning. *Physical Chemistry Chemical Physics*, 22(41):23618–23626.

Paul, S. M., Mytelka, D. S., Dunwiddie, C. T., Persinger, C. C., Munos, B. H., Lindborg, S. R., and Schacht, A. L. (2010). How to improve R&D productivity: the pharmaceutical industry's grand challenge. *Nature Reviews Drug Discovery*, 9(3):203–214.

Perola, E. and Charifson, P. S. (2004). Conformational analysis of drug-like molecules bound to proteins: an extensive study of ligand reorganization upon binding. *Journal of Medicinal Chemistry*, 47(10):2499–2510.

Podda, M., Bacciu, D., and Micheli, A. (2020). A deep generative model for fragment-based molecule generation. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108, pp.2240–2250. PMLR.

Polykovskiy, D., Zhebrak, A., Sanchez-Lengeling, B., Golovanov, S., Tatanov, O., Belyaev, S., Kurbanov, R., Artamonov, A., Aladinskiy, V., Veselov, M., Kadurin, A., Johansson, S., Chen, H., Nikolenko, S., Aspuru-Guzik, A., and Zhavoronkov, A. (2020). Molecular Sets (MOSES): a benchmarking platform for molecular generation models. *Frontiers in Pharmacology*, 11:565644.

Preuer, K., Renz, P., Unterthiner, T., Hochreiter, S., and Klambauer, G. (2018). Fréchet ChemNet Distance: a metric for generative models for molecules in drug discovery. *Journal of Chemical Information and Modeling*, 58(9):1736–1741.

Pyzer-Knapp, E. O., Suh, C., Gómez-Bombarelli, R., Aguilera-Iparraguirre, J., and Aspuru-Guzik, A. (2015). What is high-throughput virtual screening? A perspective from organic materials discovery. *Annual review of materials research*, 45(1):195–216.

Radius, U. and Breher, F. (2006). To boldly pass the metal-metal quadruple bond. *Angewandte Chemie*, 45(19):3006–3010.

Ramsundar, B., Kearnes, S., Riley, P., Webster, D., Konerding, D., and Pande, V. (2015). Massively multitask networks for drug discovery. arXiv:1502.02072.

Rappoport, D., Galvin, C. J., Zubarev, D. Y., and Aspuru-Guzik, A. (2014). Complex chemical reaction networks from heuristics-aided quantum chemistry. *Journal of Chemical Theory and Computation*, 10(3):897–907.

RDKit Team (2019). RDKit: open-source cheminformatics software. `https://www.rdkit.org/`.

Renz, P., Van Rompaey, D., Wegner, J. K., Hochreiter, S., and Klambauer, G. (2019). On failure modes in molecule generation and optimization. *Drug Discovery Today: Technologies*, 32-33:55–63.

Reymond, J.-L., Ruddigkeit, L., Blum, L., and van Deursen, R. (2012). The enumeration of chemical space. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 2(5):717–733.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32(2), pp.1278–1286. PMLR.

Rogers, D. and Hahn, M. (2010). Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754.

Röse, P. and Gasteiger, J. (1990). Automated derivation of reaction rules for the EROS 6.0 system for reaction prediction. *Analytica Chimica Acta*, 235:163–168.

Sacha, M., Błaż, M., Byrski, P., Włodarczyk-Pruszyński, P., and Jastrzębski, S. (2020). Molecule Edit Graph Attention Network: modeling chemical reactions as sequences of graph edits. In *Graph Representation Learning and Beyond (GRL+) - ICML 2020 Workshop*.

Sadowski, P., Fooshee, D., Subrahmanya, N., and Baldi, P. (2016). Synergies between quantum mechanics and machine learning in reaction prediction. *Journal of Chemical Information and Modeling*, 56(11):2125–2128.

Salatin, T. D. and Jorgensen, W. L. (1980). Computer-assisted mechanistic evaluation of organic reactions. 1. Overview. *The Journal of Organic Chemistry*, 45(11):2043–2051.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training GANs. In *Advances in Neural Information Processing Systems 29*, pp.2234–2242. Curran Associates, Inc.

Samanta, B., De, A., Jana, G., Chattaraj, P. K., Ganguly, N., and Gomez-Rodriguez, M. (2019). NeVAE: a deep generative model for molecular graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33(01), pp.1110–1117. AAAI Press.

Sanchez-Lengeling, B. and Aspuru-Guzik, A. (2018). Inverse molecular design using machine learning: generative models for matter engineering. *Science*, 361(6400):360–365.

Sato, R. (2020). A survey on the expressive power of graph neural networks. arXiv:2003.04078.

Satoh, H. and Funatsu, K. (1995). SOPHIA, a knowledge base-guided reaction prediction system – utilization of a knowledge base derived from a reaction database. *Journal of Chemical Information and Computer Sciences*, 35(1):34–44.

Sayle, R. (2013). Explicit and implicit hydrogens: taking liberties with valence. `https://nextmovesoftware.com/blog/2013/02/27/explicit-and-implicit-hydrogens-taking-liberties-with-valence/`. Accessed: 2020-12-NA.

Scannell, J. W., Blanckley, A., Boldon, H., and Warrington, B. (2012). Diagnosing the decline in pharmaceutical R&D efficiency. *Nature Reviews Drug Discovery*, 11(3):191–200.

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

Schneider, G. (2013). *De Novo Molecular Design*. Wiley-VCH Verlag GmbH & Co. KGaA.

Schneider, G., Lee, M. L., Stahl, M., and Schneider, P. (2000). De novo design of molecular architectures by evolutionary assembly of drug-derived building blocks. *Journal of Computer-Aided Molecular Design*, 14(5):487–494.

Schneider, N., Lowe, D. M., Sayle, R. A., and Landrum, G. A. (2015a). Development of a novel fingerprint for chemical reactions and its application to large-scale reaction classification and similarity. *Journal of Chemical Information and Modeling*, 55(1):39–53.

Schneider, N., Sayle, R. A., and Landrum, G. A. (2015b). Get your atoms in order–an open-source implementation of a novel and robust molecular canonicalization algorithm. *Journal of Chemical Information and Modeling*, 55(10):2111–2120.

Schütt, K. T., Kindermans, P.-J., Sauceda, H. E., Chmiela, S., Tkatchenko, A., and Müller, K.-R. (2017). SchNet: a continuous-filter convolutional neural network for modeling quantum interactions. In *Advances in Neural Information Processing Systems 30*, volume 30, pp.991–1001. Curran Associates, Inc.

Schuur, J. H., Selzer, P., and Gasteiger, J. (1996). The coding of the three-dimensional structure of molecules by molecular transforms and its application to structure-spectra correlations and studies of biological activity. *Journal of Chemical Information and Computer Sciences*, 36(2):334–344.

Schwalbe-Koda, D. and Gómez-Bombarelli, R. (2020). Generative models for automatic chemical design. In *Machine Learning Meets Quantum Physics*, pp.445–467. Springer International Publishing.

Schwaller, P., Gaudin, T., Lanyi, D., Bekas, C., and Laino, T. (2018). "Found in translation": predicting outcomes of complex organic chemistry reactions using neural sequence-to-sequence models. *Chemical Science*, 9(28):6091–6098.

Schwaller, P., Laino, T., Gaudin, T., Bolgar, P., Hunter, C. A., Bekas, C., and Lee, A. A. (2019). Molecular Transformer: a model for uncertainty-calibrated chemical reaction prediction. *ACS Central Science*, 5(9):1572–1583.

Schwaller, P., Petraglia, R., Zullo, V., Nair, V. H., Haeuselmann, R. A., Pisoni, R., Bekas, C., Iuliano, A., and Laino, T. (2020). Predicting retrosynthetic pathways using a combined linguistic model and hyper-graph exploration strategy. *Chemical Science*, 11(12):3316–3325.

Seff, A., Zhou, W., Damani, F., Doyle, A., and Adams, R. P. (2019). Discrete object generation with reversible inductive construction. In *Advances in Neural Information Processing Systems 32*, pp.10353–10363. Curran Associates, Inc.

Segler, M. H. S., Kogej, T., Tyrchan, C., and Waller, M. P. (2018a). Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS Central Science*, 4(1):120–131.

Segler, M. H. S., Preuss, M., and Waller, M. P. (2018b). Planning chemical syntheses with deep neural networks and symbolic AI. *Nature*, 555(7698):604–610.

Segler, M. H. S. and Waller, M. P. (2017a). Modelling chemical reasoning to predict and invent reactions. *Chemistry*, 23(25):6118–6128.

Segler, M. H. S. and Waller, M. P. (2017b). Neural-symbolic machine learning for retrosynthesis and reaction prediction. *Chemistry*, 23(25):5966–5971.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2016). Taking the human out of the loop: a review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. (2018). Pitfalls of graph neural network evaluation. In *Relational Representation Learning - NeurIPS 2018 Workshop*.

Sheridan, R. P. and Kearsley, S. K. (2002). Why do we need so many chemical similarity search methods? *Drug Discovery Today*, 7(17):903–911.

Shi, C., Xu, M., Guo, H., Zhang, M., and Tang, J. (2020). A graph to graphs framework for retrosynthesis prediction. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pp.8818–8827. PMLR.

Shoichet, B. K. (2004). Virtual screening of chemical libraries. *Nature*, 432(7019):862–865.

Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60.

Simm, G. N. and Reiher, M. (2017). Context-driven exploration of complex chemical reaction networks. *Journal of Chemical Theory and Computation*, 13(12):6108–6119.

Simm, G. N. C. and Hernández-Lobato, J. M. (2020). A generative model for molecular distance geometry. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pp.8949–8958. PMLR.

Simm, G. N. C., Pinsler, R., and Hernández-Lobato, J. M. (2020). Reinforcement learning for molecular design guided by quantum mechanics. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pp.8959–8969. PMLR.

Simonovsky, M. and Komodakis, N. (2018). GraphVAE: towards generation of small graphs using variational autoencoders. In *Artificial Neural Networks and Machine Learning – ICANN 2018*, pp.412–422. Springer International Publishing.

Socorro, I. M., Taylor, K., and Goodman, J. M. (2005). ROBIA: a reaction prediction program. *Organic Letters*, 7(16):3541–3544.

Somnath, V. R., Bunne, C., Coley, C. W., Krause, A., and Barzilay, R. (2020). Learning graph models for template-free retrosynthesis. In *Graph Representation Learning and Beyond (GRL+) - ICML 2020 Workshop*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, pp. 3104–3112. Curran Associates, Inc.

Szymkuć, S., Gajewska, E. P., Klucznik, T., Molga, K., Dittwald, P., Startek, M., Bajczyk, M., and Grzybowski, B. A. (2016). Computer-assisted synthetic planning: the end of the beginning. *Angewandte Chemie*, 55(20):5904–5937.

The World Bank Group (2020). World Bank open data. `https://data.worldbank.org/`. Accessed: 2020-6-NA.

Thomas, N., Smidt, T., Kearnes, S., Yang, L., Li, L., Kohlhoff, K., and Riley, P. (2018). Tensor field networks: rotation- and translation-equivariant neural networks for 3D point clouds. arXiv:1802.08219.

Todd, M. H. (2005). Computer-aided organic synthesis. *Chemical Society Reviews*, 34(3):247–266.

Todeschini, R. and Consonni, V. (2000). *Handbook of Molecular Descriptors*. WILEY-VCH Verlag GmbH.

Tolstikhin, I., Bousquet, O., Gelly, S., and Schoelkopf, B. (2018). Wasserstein auto-encoders. In *International Conference on Learning Representations 2018*.

Trinajstic, N. (2018). *Chemical Graph Theory*. Routledge.

Tripp, A., Daxberger, E., and Hernández-Lobato, J. M. (2020). Sample-efficient optimization in the latent space of deep generative models via weighted retraining. In *Advances in Neural Information Processing Systems 33 (To Appear)*, pp. 11259–11272. Curran Associates, Inc.

Ugi, I., Bauer, J., Bley, K., Dengler, A., Dietz, A., Fontain, E., Gruber, B., Herges, R., Knauer, M., Reitsam, K., and Stein, N. (1993). Computer-assisted solution of chemical problems—the historical development and the present state of the art of a new discipline of chemistry. *Angewandte Chemie, International Edition*, 32(2):201–227.

Unterthiner, T., Mayr, A., Klambauer, G., Steijaert, M., Wegner, J. K., Ceulemans, H., and Hochreiter, S. (2014). Deep learning as an opportunity in virtual screening. In *Neural Information Processing Systems Workshop on Machine Learning for Clinical Data Analysis, Healthcare and Genomics*.

van Hilten, N., Chevillard, F., and Kolb, P. (2019). Virtual compound libraries in computer-assisted drug discovery. *Journal of Chemical Information and Modeling*, 59(2):644–651.

Van Moffaert, K. and Nowe, A. (2014). Multi-objective reinforcement learning using sets of Pareto dominating policies. *Journal of Machine Learning Research*, 15:3663–3692.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pp. 5998–6008. Curran Associates, Inc.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *International Conference on Learning Representations 2018*.

Venkatraman, V., Pérez-Nueno, V. I., Mavridis, L., and Ritchie, D. W. (2010). Comprehensive comparison of ligand-based virtual screening tools against the DUD data set reveals limitations of current 3D methods. *Journal of Chemical Information and Modeling*, 50(12):2079–2093.

Vinkers, H. M., de Jonge, M. R., Daeyaert, F. F. D., Heeres, J., Koymans, L. M. H., van Lenthe, J. H., Lewi, P. J., Timmerman, H., Van Aken, K., and Janssen, P. A. J. (2003). SYNOPSIS: SYNthesize and OPtimize system in silico. *Journal of Medicinal Chemistry*, 46(13):2765–2773.

Walters, W. P. (2019). Virtual chemical libraries. *Journal of Medicinal Chemistry*, 62(3):1116–1124.

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

Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., and Zhang, Z. (2019). Deep Graph Library: a graph-centric, highly-performant package for graph neural networks. arXiv:1909.01315.

Wang, W., Arora, R., Livescu, K., and Bilmes, J. (2015). On deep multi-view representation learning. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pp.1083–1092. PMLR.

Warr, W. A. (2014). A short review of chemical reaction database systems, computer-aided synthesis design, reaction prediction and synthetic feasibility. *Molecular Informatics*, 33(6-7):469–476.

Warr, W. A. (2015). Many InChIs and quite some feat. *Journal of Computer-Aided Molecular Design*, 29(8):681–694.

Wei, J. N., Duvenaud, D., and Aspuru-Guzik, A. (2016). Neural networks for the prediction of organic chemistry reactions. *ACS Central Science*, 2(10):725–732.

Weininger, D. (1988). SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36.

Weininger, D., Weininger, A., and Weininger, J. L. (1989). SMILES. 2. Algorithm for generation of unique SMILES notation. *Journal of Chemical Information and Computer Sciences*, 29(2):97–101.

Weisfeiler, B. and Lehman, A. A. (1968). A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Tekhnicheskaya Informatsiya*, 2(9):12–16.

Wildman, S. A. and Crippen, G. M. (1999). Prediction of physicochemical parameters by atomic contributions. *Journal of Chemical Information and Computer Sciences*, 39(5):868–873.

Williams, K., Bilsland, E., Sparkes, A., Aubrey, W., Young, M., Soldatova, L. N., De Grave, K., Ramon, J., de Clare, M., Sirawaraporn, W., Oliver, S. G., and King, R. D. (2015). Cheaper faster drug development validated by the repositioning of drugs against neglected tropical diseases. *Journal of the Royal Society Interface*, 12(104):20141289.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280.

Wiswesser, W. J. (1982). How the WLN began in 1949 and how it might be in 1999. *Journal of Chemical Information and Computer Sciences*, 22(2):88–93.

Wiswesser, W. J. (1985). Historic development of chemical notations. *Journal of Chemical Information and Computer Sciences*, 25(3):258–263.

World Health Organization (2019). WHO model lists of essential medicines. `https://www.who.int/medicines/publications/essentialmedicines/en/`. Accessed: 2019-11-NA.

Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. (2018). MoleculeNet: a benchmark for molecular machine learning. *Chemical Science*, 9(2):513–530.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *International Conference on Learning Representations 2019*.

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

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., and Zhou, B. (2020). Discovery of highly potent, selective, and orally efficacious p300/CBP histone acetyltransferases inhibitors. *Journal of Medicinal Chemistry*, 63(3):1337–1360.

Yoo, S., Kim, Y.-S., Lee, K. H., Jeong, K., Choi, J., Lee, H., and Choi, Y. S. (2020). Graph-Aware Transformer: is attention all graphs need? arXiv:2006.05213.

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

You, J., Liu, B., Ying, Z., Pande, V., and Leskovec, J. (2018a). Graph Convolutional Policy Network for goal-directed molecular graph generation. In *Advances in Neural Information Processing Systems 31*, pp.6410–6421. Curran Associates, Inc.

You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. (2018b). GraphRNN: generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp.5708–5717. PMLR.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R., and Smola, A. (2017). Deep sets. In *Advances in Neural Information Processing Systems 30*, pp. 3391–3401. Curran Associates, Inc.

Zhang, M., Jiang, S., Cui, Z., Garnett, R., and Chen, Y. (2019). D-VAE: a variational autoencoder for directed acyclic graphs. In *Advances in Neural Information Processing Systems 32*, pp. 1588–1600. Curran Associates, Inc.

Zhang, Q.-Y. and Aires-de Sousa, J. (2005). Structure-based classification of chemical reactions without assignment of reaction centers. *Journal of Chemical Information and Modeling*, 45(6):1775–1783.

Zhou, Z., Kearnes, S., Li, L., Zare, R. N., and Riley, P. (2019). Optimization of molecules via deep reinforcement learning. *Scientific Reports*, 9(1):10752.

Zimmerman, P. M. (2013). Automated discovery of chemically reasonable elementary reaction steps. *Journal of Computational Chemistry*, 34(16):1385–1392.

Appendix A

Appendix for Chapter 2

This appendix clarifies additional details that were omitted from the main text of the background chapter. Specifically, in the main text we kept the description of graph neural networks (GNNs) general; here, in Section A.1, we provide details of the Gated Graph Sequence Neural Networks (Li et al., 2015) we used in the experiments. Furthermore, we also demonstrate why the 1-dimensional Weisfeiler-Lehman test cannot distinguish between the pathological pair of molecules given in Box 2.2 of the main thesis.

A.1 Graph neural networks

In Section 2.2.2 the general form of a graph neural network was specified, using generic equations for the different steps. As mentioned in that section, in the experiments we used the Gated Graph Sequence Neural Network (GGNN) proposed by Li et al. (2015). This means that we can be more specific about the particular form of the equations used. For the message passing part of this architecture, the message formation function takes the form:

$$g^m\left(\boldsymbol{h}_{v,l-1}, \left\{\left(\boldsymbol{h}_{w,l-1}, e_{(w,v)}\right), \forall\, w \in \mathcal{N}(v)\right\}\right) = \sum_{w \in \mathcal{N}(v)} \mathsf{MLP}^m_{e_{(w,v)}}\left(\boldsymbol{h}_{w,l-1}\right), \tag{A.1}$$

where $\mathsf{MLP}^m_{e_{(w,v)}}(\cdot)$ is a multi-layer perceptron (MLP)[1], one for each kind of bond type. The update function takes the form:

$$g^u\left(\boldsymbol{h}_{v,l-1}, \boldsymbol{m}_{v,l}\right) = \mathsf{GRU}^u\left(\boldsymbol{h}_{v,l-1}, \boldsymbol{m}_{v,l}\right), \tag{A.2}$$

[1] Note that here we are abusing the general terminology somewhat; often in practice we use just a single layer for the MLPs (i.e. a learnable linear transform).

where $\mathsf{GRU}^{u}(\cdot,\cdot)$ is a gated recurrent unit cell (Cho et al., 2014a).

In the readout part, the readout function takes the form of an attention weighted sum:

$$g^{r}\left(\left\{\boldsymbol{h}_{v,L}, \forall v \in \mathcal{V}\right\}\right) = \mathsf{MLP}^{\text{post}}\left(\sum_{v\in\mathcal{V}}\left[\sigma\left(\mathsf{MLP}^{\text{gate}}\left(\boldsymbol{h}_{v,L}\right)\right)\mathsf{MLP}^{\text{pre}}\left(\boldsymbol{h}_{v,L}\right)\right]\right), \tag{A.3}$$

where $\sigma(\cdot)$ is the sigmoid function, and $\mathsf{MLP}^{\text{post}}(\cdot)$, $\mathsf{MLP}^{\text{gate}}(\cdot)$, and $\mathsf{MLP}^{\text{pre}}(\cdot)$ are again multi-layer perceptrons, although often we may use only a single layer here for simplicity.

The weights and biases associated with both the MLPs and the recurrent unit used make up the learnable parameters of our model.

A.2 Molecules the 1D WL test cannot distinguish

In Box 2.2 of Chapter 2 we described how the 1-dimensional Weisfeiler-Lehman (WL) test cannot distinguish between the dicyclopropylmethane and norbornane molecules. Figure A.1 demonstrates why this is the case by running through the iterations of the WL algorithm until convergence and showing that the resulting multiset of node labels for each of the two graphs is identical.

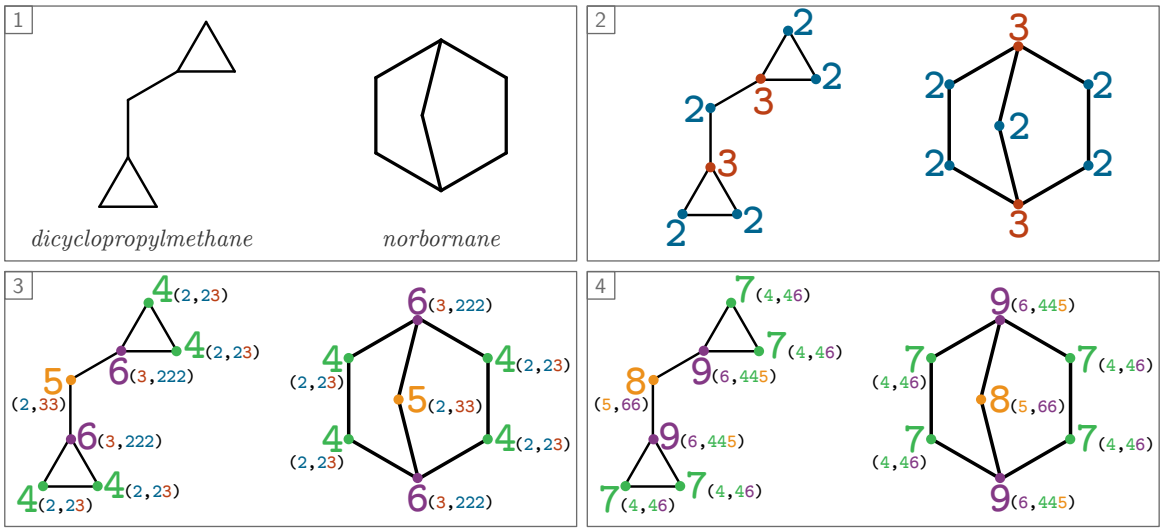


Figure A.1 *Demonstration of why the 1-dimensional Weisfeiler-Lehman (WL) test cannot distinguish between the dicyclopropylmethane and norbornane molecules (see box 1). In the 1-dimensional WL test, each node in each of the graphs is given a label (also often referred to as a "color"; shown here with the colored integers). Each label is initialized using the corresponding node's properties (e.g. its degree; see box 2). The test then goes through a series of iterations where the label for a node is updated based on its old label and the labels of its neighbors (see boxes 3 and 4 – where the parenthesis show how the new labels are derived from the old ones; for further details please refer to e.g. Morris et al., 2019, Eq. 1). These iterations continue until either the multiset of labels for each of the graphs differ (meaning that the graphs are not isomorphic) or the test converges (meaning that the graphs are potentially isomorphic). In this particular example, one can see that the test converges after 3 steps (i.e. one could relabel "7" as "4", "8" as "5", and "9" as "6" in box 4 of this figure without affecting the overall labeling in any meaningful way).*

Appendix B

Appendix for Chapter 3

This appendix clarifies additional details that were omitted from the main text of the chapter on reaction prediction (Chapter 3). In particular we include the following sections:

Section B.1, Further information on dataset preprocessing We provide further details of how we preprocess the USPTO dataset to extract approximate reaction mechanisms, supplementing Section 3.3.1 of the main text.

Section B.2, Experimental details We provide details of the specific architectures used in our experiments. We also provide additional information about the baselines used.

Section B.3, Further experimental results We provide further examples of the predictions made by our models.

B.1 Further information on dataset preprocessing

This section provides further details on how we extract reactions with linear electron flow topology from the USPTO dataset, complementing Figure 3.4 in the main text. We start from the USPTO reaction SMILES string and list of bond changes (see § 3.3.1) and from this wish to find the electron path.

The first step is to look at the bond changes present in a reaction. Each atom on the ends of the path will be involved in exactly one bond change; the atoms in the middle will be involved in two. We can then line up bond change pairs so that neighboring pairs have one atom in common, with this ordering forming a path. For instance, given the pairs `"11-13, 14-10, 10-13"`, we form the unordered path `"14-10, 10-13, 13-11"`. If we

are unable to form such a path, for instance due to two paths being present as a result of multiple reaction stages, then we discard the reaction.

For training our model we want to find the ordering of our path, so that we know in which direction the electrons flow. To do this we examine how the properties of the atoms at each end of our path change before and after the reaction. In particular, we look at the changes in charge and attached implicit hydrogen counts (as discussed in Section 2.1.1, hydrogens are not treated as separate nodes in our graph, but the number attached to a node can be inferred from the atom type, its charge and the node's associated bonds). The gain of negative charge (or analogously the gain of hydrogen as H^+ ions without changing charge) indicates that electrons have arrived at this atom, implying that this is the end of the path; vice versa for the start of the path. However, sometimes the difference in atom properties before and after a reaction is not available in the USPTO data. This is because unfortunately only major products are recorded, and so details of what happens to some of the reactant molecules' atoms may be missing. In these cases we fall back to using an element's "electronegativity" to estimate the direction of our path, with more electronegative atoms attracting electrons towards them and so being at the end of the path.

The next step of filtering checks that the path alternates between add steps and remove steps. This is done by analyzing and comparing the bond changes on the path in the reactant and product molecules. Reactions that involve greater than one change between atoms (for instance going from no bond between two atoms in the reactants to a double bond between the two in the products) can indicate multi-stage or other kinds of non-LEF reactions, and so are discarded. As a final validation, we use RDKit to produce all the intermediate and final products induced by our path acting on the reactants, to confirm that the final product that is produced by our extracted electron path is consistent with the major product SMILES in the USPTO dataset.

B.2 Experimental details

In this section we provide specific details about the architectures of our models used in the experiments; further details can also be found in our code, `https://github.com/john-bradshaw/electro`. At the end of this section, we provide additional information about the implementations of the baselines we compared against.

Table B.1 *Atom features we use as input to the GGNN. These are calculated using RDKit (RDKit Team, 2019).*

Feature	Description
Atom type	One hot (72 possible elements in total)
Degree	One hot (0, 1, 2, 3, 4, 5, 6, 7, 10)
Explicit Valence	One hot (0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14)
Hybridization	One hot (SP, SP2, SP3, other)
H count	Integer
Electronegativity	Float
Atomic number	Integer
Part of an aromatic ring	Boolean

Table B.2 *The GNNs $\mathsf{GNN}^{reagents}(\cdot)$ and $\mathsf{GNN}^{cont}(\cdot)$ have different readout architectures. We provide details of the dimensionalities of the vector spaces the MLPs operate with in this table (see Eq. A.3).*

	$\mathsf{GNN}^{\text{reagents}}(\cdot)$	$\mathsf{GNN}^{\text{cont}}(\cdot)$
$\mathsf{MLP}^{\text{gate}}(\cdot)$	$\mathbb{R}^{101} \rightarrow \mathbb{R}^{1}$	$\mathbb{R}^{101} \rightarrow \mathbb{R}^{1}$
$\mathsf{MLP}^{\text{pre}}(\cdot)$	$\mathbb{R}^{101} \rightarrow \mathbb{R}^{202}$	$\mathbb{R}^{101} \rightarrow \mathbb{R}^{202}$
$\mathsf{MLP}^{\text{post}}(\cdot)$	$\mathbb{R}^{202} \rightarrow \mathbb{R}^{100}$	$\mathbb{R}^{202} \rightarrow \mathbb{R}^{1}$

B.2.1 Architecture and training details

In this section we provide further details of our model architectures.

Graph neural networks (GNNs) As described in Section 3.2 (in particular see Alg. 3.1), we used three GNNs: $\mathsf{GNN}^{\text{node}}(\cdot)$ for computing node embeddings (i.e. without a readout part); $\mathsf{GNN}^{\text{reagents}}(\cdot)$ for computing a molecule-level embedding of the reagents; and finally $\mathsf{GNN}^{\text{cont}}(\cdot)$ for computing the continuation logit. All three of these networks are based around the GGNN architecture (§ A.1).

For the message passing part (i.e. Eq. A.1 and Eq. A.2), these GNNs share the same architecture (including weights). Here, we run four stages of messages passing (i.e. $L = 4$). The GNNs use as initial node attributes, $\mathbf{x}_v$, the atom features shown in Table B.1. In our implementation, the node-embeddings have a dimensionality of 101.

The $\mathsf{GNN}^{\text{reagents}}(\cdot)$ and $\mathsf{GNN}^{\text{cont}}(\cdot)$ networks have an additional readout part (Eq. A.3). These two GNNs use different MLPs here, projecting to graph-level embeddings with different dimensionalities. We provide further details of this in Table B.2.

Action distributions Section 3.2 of our thesis discusses the different components of our model. To compute the atoms on the path we are interested in computing two conditional probability terms: (a) $p_\theta^{\text{start}}(a_0|\mathcal{M}_0, \mathcal{M}_e)$, the probability of the initial state, a_0, given the reactants and reagents; and (b) $p_\theta(a_t|\mathcal{M}_t, a_{t-1}, t)$, the conditional probability of the next state, a_t, given the intermediate products, $\mathcal{M}_t$, for $t > 0$.

These distributions are parameterized by NNs (neural networks): $f^{\text{add}}(\cdot,\cdot)$, $f^{\text{remove}}(\cdot,\cdot)$, and $f^{\text{start}}(\cdot,\cdot)$. All of these operate on node embeddings created by the same GNN. In this section we shall go through each of these networks in turn.

The networks for $f^{\text{add}}(\cdot,\cdot)$ and $f^{\text{remove}}(\cdot,\cdot)$, that operate on each node to produce an action logit, are both feedforward NNs consisting of one hidden layer of 100 units. Concatenated onto the node features going into these networks are the node features belonging to the previous atom on the path. To write this out more explicitly for the add network:

$$f^{\text{add}}\left(\boldsymbol{H}_{\mathcal{M}_t}, a_{t-1}\right)_v = \mathsf{MLP}^{\text{add}}\left(\boldsymbol{h_v} \parallel \boldsymbol{h_{a_{t-1}}}\right), \tag{B.1}$$

where v indexes a particular atom, $\boldsymbol{h_v}$ denotes the final node embedding from $\mathsf{GNN}^{\text{node}}(\cdot)$ for atom v (i.e. a particular row of $\boldsymbol{H}_{\mathcal{M}_t}$), "$\parallel$" the concatenation operator, and $\mathsf{MLP}^{\text{add}}(\cdot)$ denotes a regular feedforward neural network. The network for $f^{\text{remove}}(\cdot,\cdot)$ takes a similar form.

The final function, $f^{\text{start}}(\cdot,\cdot)$, is represented by a NN with hidden layers of 100 units. When conditioning on reagents (i.e. for ELECTRO) the reagent embeddings calculated by $\mathsf{GNN}^{\text{reagents}}(\cdot)$ are concatenated onto the node embeddings and we use two hidden layers for our NN. When ignoring reagents (i.e for ELECTRO-LITE) we use one hidden layer for this network. To write this out more explicitly:

$$f^{\text{start}}\left(\boldsymbol{H}_{\mathcal{M}_0}, \boldsymbol{m}_{\mathcal{M}_e}\right)_v = \mathsf{MLP}^{\text{start}}\left(\boldsymbol{h_v} \parallel \boldsymbol{m}_{\mathcal{M}_e}\right), \tag{B.2}$$

where again v indexes a particular atom, $\boldsymbol{h_v}$ denotes the node embedding from $\mathsf{GNN}^{\text{node}}(\cdot)$ for atom v (i.e. a particular row of $\boldsymbol{H}_{\mathcal{M}_0}$), $\boldsymbol{m}_{\mathcal{M}_e}$ denotes the graph-level embedding from $\mathsf{GNN}^{\text{reagents}}(\cdot)$, and $\mathsf{MLP}^{\text{start}}(\cdot)$ denotes a regular feedforward neural network.

Although we found choosing the first entry in the electron path is often the most challenging decision, and greatly benefits from reagent information, we also considered a version of ELECTRO where we fed in the reagent information at every step. In other words, the networks for $f^{\text{add}}(\cdot,\cdot)$ and $f^{\text{remove}}(\cdot,\cdot)$ also received the reagent embeddings calculated by $\mathsf{GNN}^{\text{reagents}}(\cdot)$ concatenated onto their inputs. On the mechanism prediction task (Table 3.2), this new model gets a slightly improved top-1 accuracy of 78.4% (77.8%

before) but a similar top-5 accuracy of 94.6% (94.7% before). On the reaction product prediction task (Table 3.3), the new model gets 87.5%, 94.4%, and 96.0% top-1, 3, and 5 accuracies (87.0%, 94.5%, and 95.9% before). The trade-off is this new model is somewhat more complicated and requires a greater number of parameters.

Training We train everything using the Adam optimizer (Kingma and Ba, 2014) and an initial learning rate of 0.0001, which we decay after 5 and 9 epochs by a factor of 0.1. We train for a total of 10 epochs. For training we use reaction minibatch sizes of one, although these can consist of multiple intermediate graphs.

B.2.2 Baselines

For the baselines in Table 3.3 we used the following implementations:

WLDN (Jin et al., 2017): We used the official implementation at:
`https://github.com/wengong-jin/nips17-rexgen`
We used the pre-trained model available in this online repository for the FTS comparison.

Seq2Seq (Schwaller et al., 2018): Unfortunately, no "official" implementation was available so we used an implementation of a sequence-to-sequence model from the OpenNMT library (Klein et al., 2017). We matched the hyperparameters, architecture details, and tokenization process to that reported in Schwaller et al. (2018), and trained the model up from scratch.

B.3 Further experimental results

This section provides further examples of the paths predicted by our model. In Figures B.1 and B.2 we wish to show how the model has learned chemical trends by testing it on additional textbook reactions (see also Figure 3.5 in the main text).

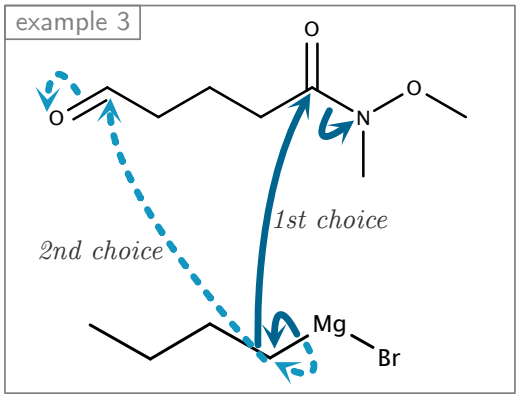


Figure B.1 *Predicted mechanism of our model on reactant molecules. The dark blue solid (—) arrows show the model's preferred mechanism, whereas lighter blue dashed (−−) arrows show the model's second choice. Here, the first-choice prediction is incorrect, but chemically reasonable, as the Weinreb amide is typically used together in reactions with magnesium species. The second-choice prediction is correct.*

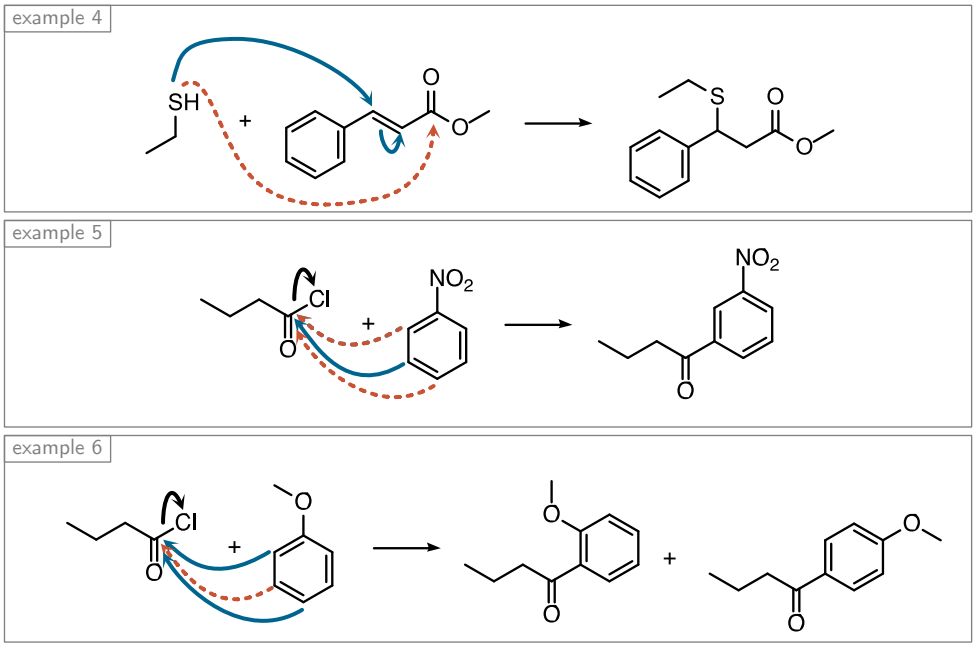


Figure B.2 *Additional typical selectivity examples. Here, the expected product is shown on the right. The dark blue solid (—) arrows indicate the top ranked paths from our model, the red dashed (−−) arrows indicate other possibly competing but incorrect steps, which the model does not predict to be of high probability. In all cases, our model predicted the correct products. In examples 5 and 6, our model correctly recovers the regioselectivity expected in electrophilic aromatic substitutions.*

Appendix C

Appendix for Chapter 4

This appendix clarifies additional details that were omitted from the main text of the chapter introducing MOLECULE CHEF (Chapter 4). In particular we include the following sections:

Section C.1, Dataset details We provide further information on the dataset we used in the evaluation.

Section C.2, Experimental details We provide details of the specific architectures used in our experiments, details of the implementations we used for the baselines in the generation section, as well as a short commentary on why we used the Molecular Transformer as the reaction prediction component.

Section C.3, Further random walk examples We provide further examples of random walks in the latent space of MOLECULE CHEF and describe the rationales used for the expert labels.

Section C.4, Further experiments on retrosynthesis We provide further experiments evaluating the use of MOLECULE CHEF's latent space for retrosynthesis.

C.1 Dataset details

In this section we provide further details about the molecules used in training our model and the baselines. We also describe details of the molecules used in the retrosynthesis experiments.

For MOLECULE CHEF's building block vocabulary we use reactants that occur at least 15 times in the USPTO training dataset, as processed and split by Jin et al. (2017). This

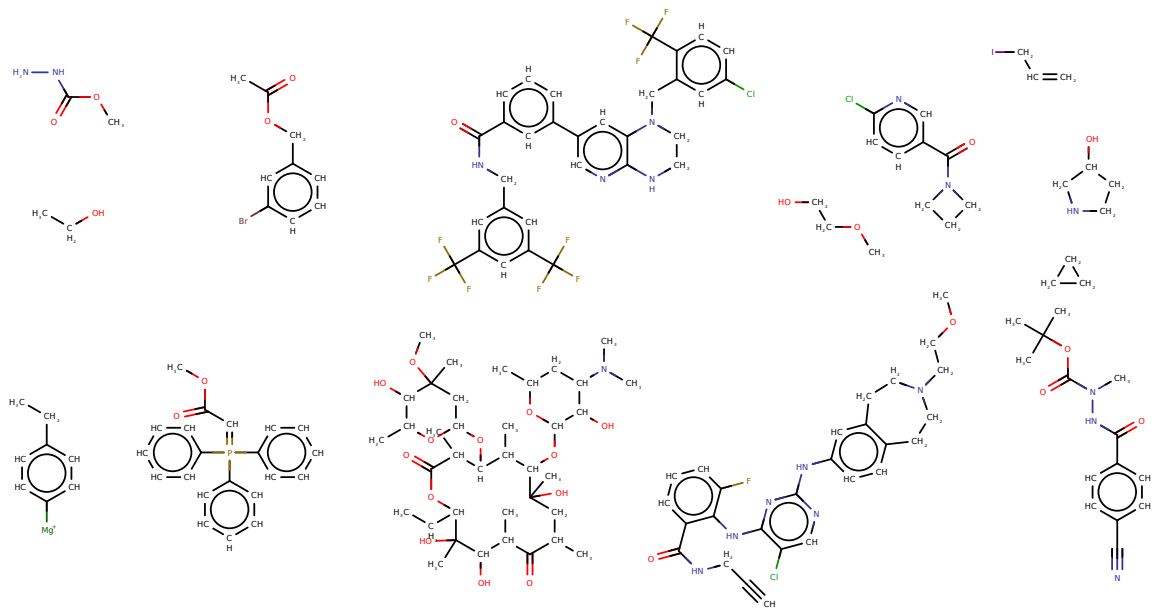


Figure C.1 *Examples of molecules found in the dataset we use for training the baselines. This is a subset of the molecules found in USPTO (Lowe, 2012). The dataset for training the baselines consists of the building blocks that* MOLECULE CHEF *can select along with the associated products. It contains complex molecules with challenging structures to learn.*

dataset uses reactions collected by Lowe (2012) from the US patents. In total we have 4,344 building block molecules (possible reactants), and a training set of 34,426 unique reactant bags for which these building blocks co-occur. Each reactant bag is associated with a product.

We train the baselines on these building block molecules and the associated products. This results in a dataset of approximately 37,000 unique molecules, containing a wide variety of heavy atoms:

{ ‘‘Al’’, ‘‘B’’, ‘‘Br’’, ‘‘C’’, ‘‘Cl’’, ‘‘Cr’’, ‘‘Cu’’, ‘‘F’’, ‘‘I’’, ‘‘K’’, ‘‘Li’’, ‘‘Mg’’, ‘‘Mn’’, ‘‘N’’, ‘‘Na’’, ‘‘O’’, ‘‘P’’, ‘‘S’’, ‘‘Se’’, ‘‘Si’’, ‘‘Sn’’, ‘‘Zn’’ } .

Some examples of the molecules found in the dataset are shown in Figure C.1. Note that the dataset is challenging to model, especially compared to some of the more common benchmark datasets used elsewhere (such as ZINC, Irwin et al., 2012), due to its small overall size and the large number of heavy atoms present.

We use examples from the USPTO test dataset when performing the retrosynthesis experiments. However, we first filter out any reactions where the exact same reactan-

t/product pair is also present in MOLECULE CHEF's training data[1]. Then we split the resultant dataset into two subsets. The first, which we refer to as the "reachable" dataset, contains only reactant molecules in MOLECULE CHEF's vocabulary of possible building blocks. The second, which we refer to as the "unreachable" dataset, contains reactions with at least one reactant not in the vocabulary.

C.2 Experimental details

In this section we provide specific details about the architectures of the models used in our experiments. Further details can also be found in our code, `https://github.com/john-bradshaw/molecule-chef`. Following a similar layout to Section B.2 in the previous appendix, we also provide additional information about the implementations of the baselines we compared against. In the final part of this section we provide further details on why we picked the Molecular Transformer for reaction prediction.

C.2.1 Architecture and training details

Here we give a brief description of the architecture of MOLECULE CHEF used in the experiments.

Computing vector embeddings of molecular graphs using graph neural networks Both MOLECULE CHEF's encoder and decoder networks rely on vector representations of molecules (i.e. molecule-level embeddings) computed by a graph neural network (GNN; § 2.2.2). For computing the vector representations of molecular graphs we again used Gated Graph Sequence Neural Networks (§ A.1), with the same network shared in both the encoder and decoder. We ran four stages of message passing (i.e. $L = 4$) and the node embeddings had a dimension of 101. We initialized the node embeddings with the same atom features we used for ELECTRO, shown in Table B.1. The final molecule-level embeddings had a dimension of 50.

Encoder MOLECULE CHEF's encoder takes in a multiset of reactants and outputs the parameters of a Gaussian distribution over $\boldsymbol{z}$.

The encoder sums the vector embeddings of the molecules present in the reactant multiset (each computed using a GNN) to get a 50 dimensional vector embedding of the

[1] After canonicalization and the removal of reagents, the USPTO train and test dataset has some reactions present in both sets.

Table C.1 *Parameters for the GRU used in the decoder*

Parameter	Value
GRU hidden size	50
GRU number of layers	2
GRU maximum number of steps	5

entire multiset. This embedding is fed through a single hidden layer NN (neural network), with a hidden layer size of 200, to parameterize the mean and diagonal of the covariance matrix of a 25 dimensional multivariate Gaussian distribution over $\boldsymbol{z}$.

Decoder The decoder maps from the latent space, $\boldsymbol{z}$, to a multiset of reactants, $\boldsymbol{x}$, selected from a vocabulary of possible building blocks. It does this through a sequential process, selecting one reactant at a time using a gated recurrent unit (GRU; Cho et al., 2014a) RNN (recurrent neural network). The hyperparameters used for specifying the architecture of this GRU are shown in Table C.1. The initial hidden state of the RNN is set using the result from a learned linear projection of $\boldsymbol{z}$. The output of the GRU at each time step is fed through a single hidden layer NN (with a hidden size of 128) to form an output vector. The dot product of this output vector is formed with the HALT embedding and also with each of the possible building blocks' embeddings to form logits for the next reactant output of the decoder. The embedding of the selected reactant is fed back in as input for the RNN at the next step.

Property predictor In Section 4.2.2 of the main text we discussed how, while training the WAE, we can also train a property predictor from the latent space to a property of interest such as the QED score (§ 4.3.2). To this end, we used a fully connected NN (neural network) with two hidden layers (both with a dimensionality of 40) for the QED score property predictor. The loss from this network was added to the WAE loss when training the model for the local optimization and retrosynthesis tasks.

Training We trained the WAE (and property predictor when applicable) for 100 epochs. We used the Adam optimizer (Kingma and Ba, 2014), with an initial learning rate of 0.001. We decayed the learning rate by a factor of 10 every 40 epochs.

C.2.2 Baselines

Further details of the baselines in Table 4.1 are shown below:

CGVAE (Liu et al., 2018): We used the official implementation:
`https://github.com/microsoft/constrained-graph-variational-autoencoder`
We decided to include element-charge-valence triplets that occur at least 10 times over all of the molecules in the training data. At generation time we pick one starting node at random.

SMILES LSTM (Segler et al., 2018a) : We used the implementation available at:
`https://github.com/BenevolentAI/guacamol_baselines`
This implementation follows Segler et al. (2018a), which has as its alphabet a list of all individual element symbols, plus special characters used in SMILES strings. This differs from the alphabet used by the decoder in the Molecular Transformer (Schwaller et al., 2019), which instead extracts "bracketed" atoms directly from the training set using a regular expression. This means that a portion of a SMILES string such as "`[OH+]`" or "`[NH2-]`" would be represented as a single tokenized character, rather than as a sequence of five individual tokenized characters. Effectively, this makes the trade-off of increasing the alphabet size (from 47 to 203 items), while reducing the chance of making syntax errors or suggesting invalid charges. In practice, we found very little qualitative or quantitative difference in the performance of the LSTM model for the two alphabets; for sake of consistency with MOLECULE CHEF, we report the baseline using the larger alphabet.

AAE (Kadurin et al., 2017; Polykovskiy et al., 2020): We used the implementation available at:
`https://github.com/molecularsets/moses/tree/master/moses/aae`

GVAE (Kusner et al., 2017): We used the official implementation available at:
`https://github.com/mkusner/grammarVAE`

CVAE (Gómez-Bombarelli et al., 2018): We used the version available at:
`https://github.com/mkusner/grammarVAE`

C.2.3 Rationale behind choosing the Molecular Transformer

In this subsection we provide a short commentary on how we ended up using the Molecular Transformer as the reaction predictor "oracle" in Chapter 4. We hope that this provides the reader additional insight into the challenges in using/designing the models described in this thesis, as well as pointing out fruitful possible areas for future work.

Perhaps an obvious question to ask when reading Chapter 4, is why did we not use ELECTRO for the reaction prediction component of MOLECULE CHEF? Initially, we tried

using it in place of the Molecular Transformer but we found that it gave worse performance. On investigation, we found that ELECTRO would sometimes choose not to transform the reactants at all, leaving them unchanged. This would lead to poorer novelty scores. We hypothesize that this behavior could partly be due to ELECTRO not being able to predict non-LEF reactions, which might be necessary to model many of the chosen reactant bags' interactions.

We believe the current limitations of ELECTRO in this setting motivate exciting future work. One such area is the challenge of extending ELECTRO to model non-LEF reactions, which we previously discussed in Section 3.4.1. Another possible direction is to study the generalizability of reaction predictors more broadly. For the most part, reaction predictors are usually evaluated on independently and identically distributed train and test sets. Generative models of molecules built around virtual reaction schemes provide a setting (particularly when these models are being used for optimization) where this assumption may no longer hold true. It would be interesting to see how the different methods for reaction prediction compare in this setting.

C.3 Further random walk examples

In this section we provide more examples of random walks done in MOLECULE CHEF's latent space (see Figures C.3 and C.4). We include also a blown up version of Figure 4.5 from the main text (see Figure C.2). We also provide further details of the labeling scheme used in Section 4.3.1.

Rationales for expert labels in Figure 4.5 in the main text (§ 4.3.1) Denoted using letters for the rows and numbers for the columns. A3: unstable, enol; A5: unstable, aminal; B2: reactive, radical; B4: unstable ring system; B5: toxic, reactive sulfur-chloride bond, unstable ring system; B6: unstable ring system; B7: unstable ring system; B9: toxic: thioketone.

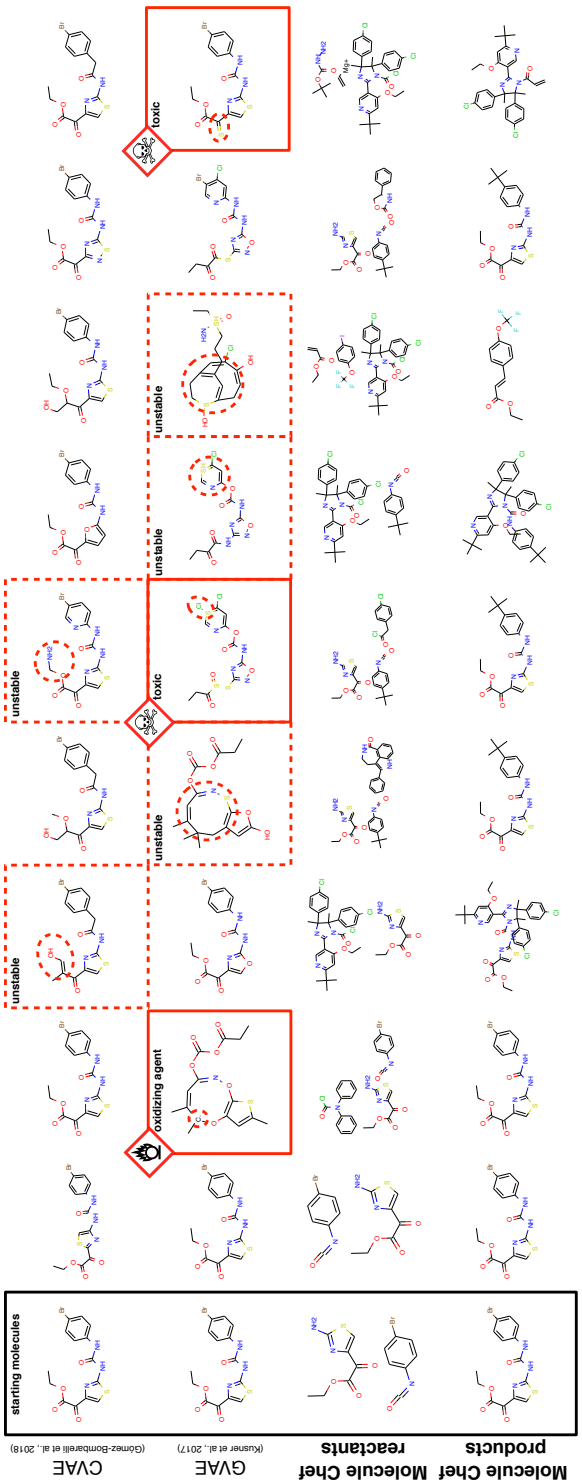


Figure C.2 *We repeat here a larger version of Figure 4.5 to allow for easier viewing. The rationales for the labels are (using letters for rows and numbers for columns – when viewed in landscape): A3: unstable, enol; A5: unstable, aminal; B2: reactive, radical; B4: unstable ring system; B5: toxic, reactive sulfur-chloride bond, unstable ring system; B6: unstable ring system; B7: unstable ring system; B9: toxic: thioketone.*

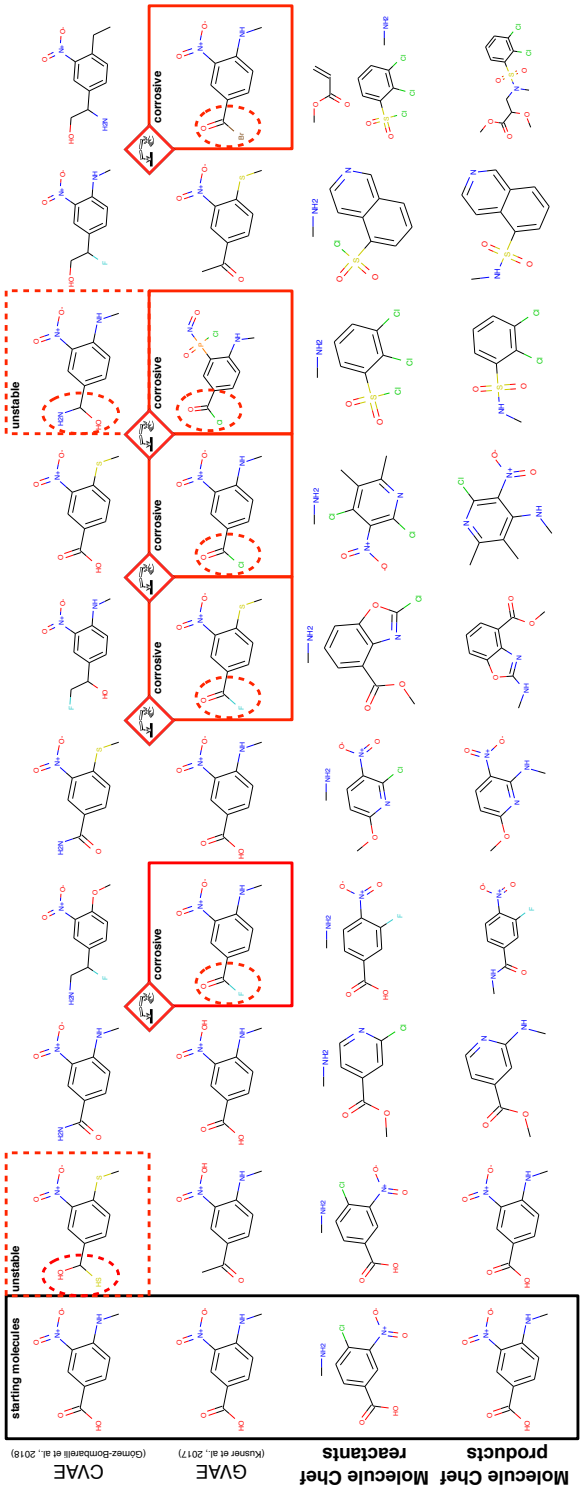


Figure C.3 *Another example random walk in latent space. See Section 4.3.1 for further details. The rationales for the labels are (using letters for rows and numbers for columns – when viewed in landscape): A1: unstable, gemthiolol; A7: unstable, gem-aminohydroxyl; B3: corrosive, acyl fluoride B5: corrosive, acyl fluoride; B6: corrosive, acyl chloride; B7: corrosive, acyl chloride; B9: corrosive, acyl bromide.*

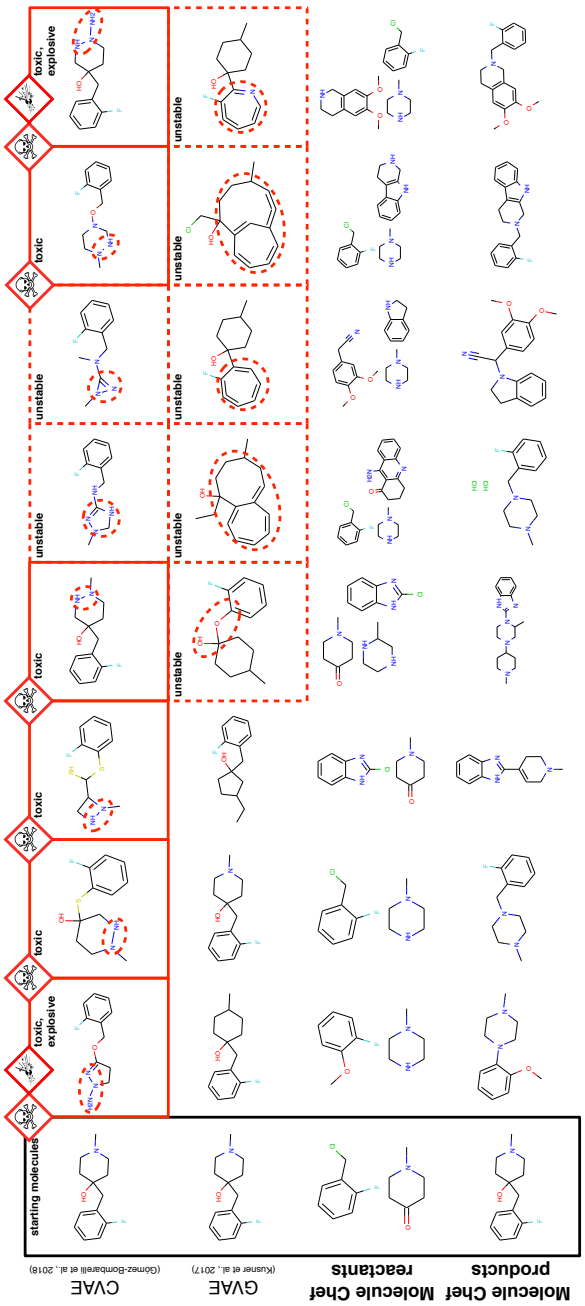


Figure C.4 *Another example random walk in latent space. See Section 4.3.1 for further details. The rationales for the labels are (using letters for rows and numbers for columns – when viewed in landscape): A1: toxic, explosive, hydrazine, three consecutive aliphatic nitrogen atoms; A2: toxic, hydrazine; A3: toxic, unstable, hydrazine, hemithioacetal; A4: toxic, hydrazine; A5: unstable, could be oxidized to 1,3,4-triazol, potentially also toxic due to N-N bond/hydrazine; A6: unstable, three-membered ring is antiaromatic; A7: toxic, hydrazine; A8: toxic, explosive, hydrazine, three consecutive aliphatic nitrogen atoms. B4: unstable, hemiacetal; B5-B8: unstable, unfavorable ring systems.*

C.4 Further experiments on retrosynthesis

In this section we first provide more retrosynthesis examples before also describing an extra experiment in which we try to assess how well the retrosynthesis pipeline is at finding molecules with similar properties to a target molecule, even if not reconstructing the correct reactants themselves.

C.4.1 Additional examples

Figures C.5 and C.6 show further individual examples of retrosynthesis suggestions made by our model (see § 4.3.3).

C.4.2 ChemNet distances between products and their reconstructions

In this section we consider an additional retrosynthesis experiment to complement the one carried out in Section 4.3.3 (see also Figure 4.10). Instead of comparing the QED score of the product and the reconstructed product, we consider looking at the differences of the products' ChemNet[2] embeddings. ChemNet embeddings are used when calculating the FCD score (Preuer et al., 2018) between molecule distributions, and so hopefully capture various properties of the molecule (Mayr et al., 2018).

In particular, we consider the *Euclidean distance* between the ChemNet embeddings of the product and reconstructed product. In order to ensure MOLECULE CHEF's latent space is structured according to these embeddings, when learning MOLECULE CHEF for this task, we include a NN that maps from the latent space to the space of ChemNet embeddings. The MSE (mean squared error) loss from this network is added to the WAE loss and is minimized during training (in a similar manner to that described in Sections 4.2.2 and 4.3.2 for the QED scores). As a baseline, we compare against the Euclidean distances between ChemNet embeddings of randomly chosen pairs of molecules from the dataset used to train the baseline models in the generation task.

The distributions of these Euclidean distances is shown in Figure C.7. Encouragingly, we see that the distribution of the distance between the product and its reconstruction has greater mass on smaller values compared to the distribution associated with the random pairs baseline.

[2]Note that this network is different to Kireev (1995)'s "ChemNet" that we discussed in Box 2.2.

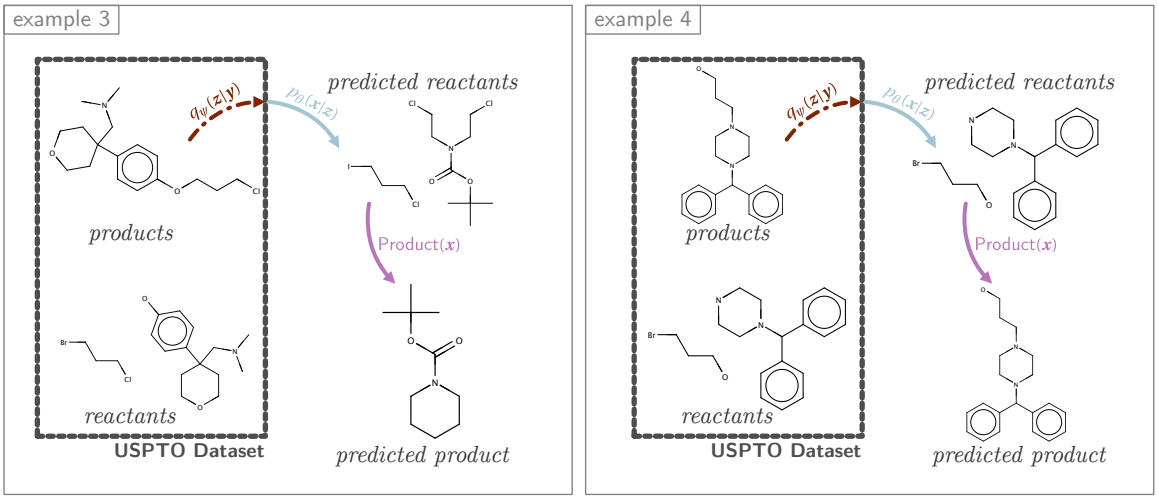


Figure C.5 *Further examples of the predicted reactants associated with a given product for product molecules not in* MOLECULE CHEF*'s training dataset; however, with reactants belonging to* MOLECULE CHEF*'s vocabulary of building blocks (i.e. "reachable" dataset).*

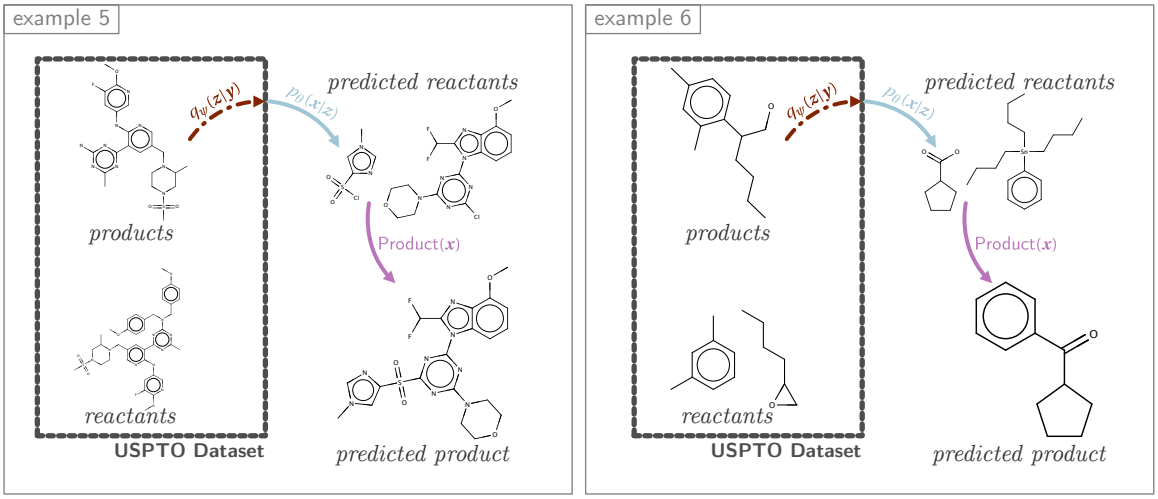
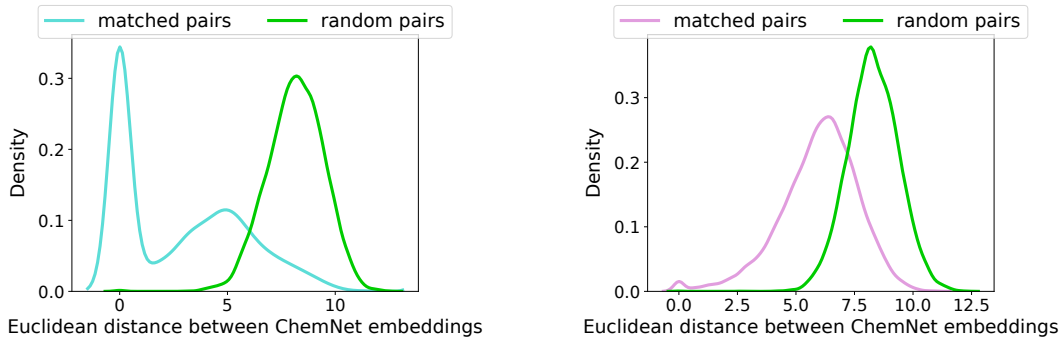


Figure C.6 *Further examples of the predicted reactants associated with a given product for product molecules not in* MOLECULE CHEF*'s training dataset, with at least one reactant not part of* MOLECULE CHEF*'s vocabulary of building blocks (i.e. "unreachable" dataset).*



(a) *When evaluated on the portion of USPTO test set reactions for which all reactants are present in* MOLECULE CHEF*'s building block vocabulary (i.e. "reachable" dataset).*

(b) *When evaluated on the portion of USPTO test set reactions for which at least one reactant is not present in* MOLECULE CHEF*'s building block vocabulary (i.e. "unreachable" dataset).*

Figure C.7 *KDE (kernel density estimate) plot showing the distribution of the Euclidean distances between the ChemNet embeddings (Mayr et al., 2018; Preuer et al., 2018) of our product and reconstructed product compared to randomly chosen molecule pairs.*

Appendix D

Appendix for Chapter 5

This appendix clarifies additional details that were omitted from the main text of the chapter introducing DoG-Gen (Chapter 5). In particular, following roughly the structure of the previous appendix, we include the following sections:

Section D.1, Dataset details We provide further information on the dataset we used in the evaluation.

Section D.2, Further model details We provide additional information about our model.

Section D.3, Synthesizability score We define the synthesizability score.

Section D.4, Experimental details We provide additional information about our experimental setup, with a more detailed description of the hyperparameters used. We also provide details of the implementations we used for the baselines in the generation task as well as a short discussion about why we used the fine-tuning approach in the optimization tasks.

Section D.5, Additional experiments We provide additional results from the evaluation described in the main text as well as a comparison to an older reaction-driven de novo design algorithm. We also provide an investigation into using DoG-Gen for synthesis planning.

D.1 Dataset details

In this section we describe how we create a dataset of synthesis DAGs, with a high-level illustration of the process given in Figure D.1.

The creation of our synthesis DAG dataset starts from the same processed and cleaned version of the USPTO dataset provided by Jin et al. (2017, § 4) we have previously described (§3.3.1)[1]. Similar to ELECTRO, we again filter out reagents (molecules that do not contribute any atoms to the final product). In addition, for creating a dataset of synthesis DAGs we also filter out multiple product reactions (97% of the dataset is already single product reactions) using the approach of Schwaller et al. (2018, § 3.1).

This processed reaction data is then used to create a "reaction network" (Grzybowski et al., 2009; Jacob and Lapkin, 2018; Segler and Waller, 2017a). To be more specific, we start with the reactant building blocks specified in Chapter 4 (for MOLECULE CHEF) as initial molecule nodes in our network. We then iterate through our list of processed reactions, adding any reactions (and the associated product molecules) (i) that depend only on molecule nodes that are already in our network, and (ii) where the product is not an initial building block. This process repeats until we can no longer add any of our remaining reactions.

This reaction network is then used to create one synthesis DAG for each molecule. To this end, starting from each possible (non building block) molecule node in our reaction network, we step backwards through the network until we find a subgraph of the reaction network (without any loops) with initial nodes that are from our collection of building blocks. When there are multiple possible routes, we pick one. This leaves us with a dataset of 72,008 synthesis DAGs, which we use approximately 90% of as training data and split the remainder into a validation dataset (of 3,601 synthesis DAGs) and a test dataset (of 3,599 synthesis DAGs).

The training set DAGs have an average of 4.6 nodes, with the final molecules containing an average of 20.5 heavy atoms and 21.7 bonds (between heavy atoms). The average number of actions required to construct these DAGs is 11, as each reactant often contributes several atoms and bonds to the product.

D.2 Further model details

In this section we provide more details on our generative model for synthesis DAGs, including pseudocode for the full generative process.

Notation We first recap the notation that we used in Chapter 5. We formally represent the DAG, $\mathfrak{D}$, as a sequence of actions, with $\mathfrak{D} = [V_1, V_2, V_3, \ldots, V_J]$. Alongside this we denote the associated action types as $\mathsf{T} = [\top_1, \top_2, \top_3, \ldots, \top_J]$. The action type entries, $\top_j$, take

[1] Here though we no longer use Jin et al. (2017)'s splits.

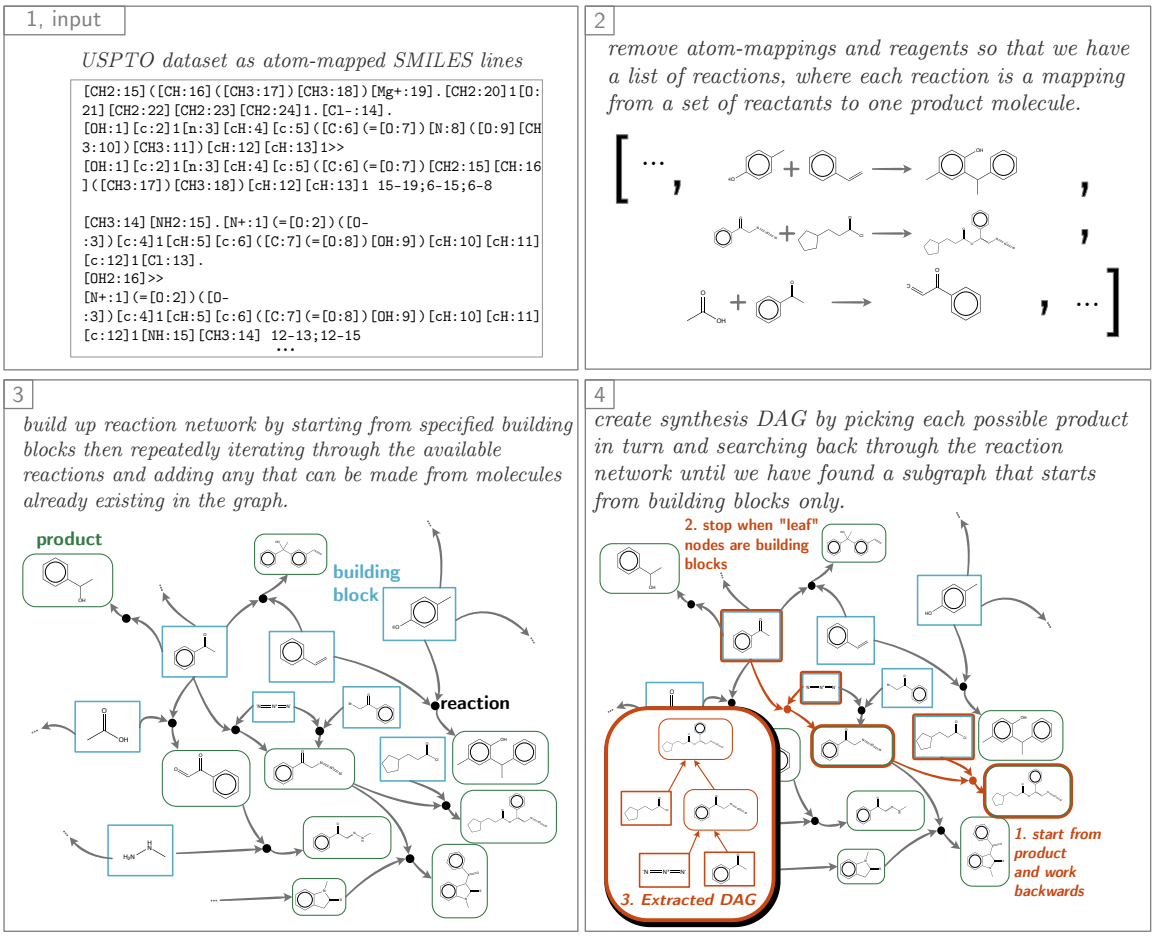


Figure D.1 *An illustration of how we create a dataset of synthesis DAGs from a dataset of reactions. We first clean up the reaction dataset by removing reagents (molecules which do not contribute atoms to the final product) and any reactions which lead to more than one product. We then form a modified reaction network (we do not allow loops back to building block molecules), which is a directed graph showing how molecules are linked to others through reactions. This process starts by adding molecule nodes corresponding to our initial building blocks. We then repeatedly iterate through our list of reactions and gradually add reaction nodes (and their associated product nodes) to the graph if both (i) the corresponding reaction's reactants are a subset of the molecule nodes already present in the graph, and (ii) the product is not a building block. Finally, for each possible product node, we iterate back through the directed edges until we have selected a subgraph without any loops, and where the initial nodes are members of our set of building blocks.*

values in $\{A1, A2, A3\}$, corresponding to the three action types. The action type entry at a particular step, $\top_j$, is fully defined by the actions (and action types) chosen previously to this time j, the exact details of which we shall come back to later. Finally the set of molecules existing in the DAG at time j are denoted (in an abuse of our notation) by $\mathcal{M}_{<j}$.

Actions and the values that they can take We now describe the potential values that the actions can take. These depend on the corresponding action type, and we denote this conditioning as $V_j^{|\top_j}$. For example, for node addition actions (i.e. $\top_j = \text{A1}$), the possible values of V_j (i.e. $V_j^{|\text{A1}}$) are either 'B' for creating a new building block node, or 'P' for a new product node. Building block actions (i.e. $\top_j = \text{A2}$) have corresponding values $V_j \in \mathcal{B}$, which determine which building block becomes a new "leaf" node in the DAG. Connectivity choice actions (i.e. $\top_j = \text{A3}$) have values $V_j \in \mathcal{M}_{<j} \cup \{\not\rightarrow_{\mathcal{I}}, \not\rightarrow_{\mathcal{F}}\}$, where $\mathcal{M}_{<j}$ denotes the current set of all molecules present in the DAG; selecting one of these molecules adds an edge from it into the new product node. The symbol $\not\rightarrow_{\mathcal{I}}$ is an intermediate product stop symbol, indicating that the new product node has been connected to all its reactants (i.e. an intermediate product has been formed); the symbol $\not\rightarrow_{\mathcal{F}}$ is a final stop symbol, which triggers production of the final product and the completion of the generative process.

As hinted at earlier, the action type, $\top_j$, is defined by the previous actions $V_1, \ldots, V_{j-1}$ and action types (see also Figure 5.3 in the main text). More specifically, this happens as follows:

- If $\top_{j-1} = \text{A1}$, then:
  - If $V_{j-1} = \text{'B'}$, then the next action type is building block selection, $\top_j = \text{A2}$.
  - If $V_{j-1} = \text{'P'}$, then the next action type is connectivity choice, $\top_j = \text{A3}$, to work out what to connect up to the product node previously selected.
- If $\top_{j-1} = \text{A2}$, then the next action type is again node addition, $\top_j = \text{A1}$ (as you will have selected a building block on the previous step).
- If $\top_{j-1} = \text{A3}$, then:
  - If $V_{j-1} \in \mathcal{M}_{<j}$, then connectivity choice continues, i.e. $\top_j = \text{A3}$.
  - If $V_{j-1} = \not\rightarrow_{\mathcal{I}}$, then the next action type is to choose a new node again, i.e. $\top_j = \text{A1}$.
  - If $V_{j-1} = \not\rightarrow_{\mathcal{F}}$, the generation is finished.

Our generative process over these actions Our model is shown at a high-level in Figure D.2 (this is an expanded version of Figure 5.4 in Chapter 5), which serves to provide an intuitive understanding of the generative process. The overall structure of the probabilistic model is rather complex, as it depends on a series of branching conditions: we therefore give pseudocode for the entire generative procedure in detail in Algorithm D.1. Much like

our generative procedure for ELECTRO (Alg. 3.1), we can interpret this as a probabilistic program; running it forward will sample from the generative process, but it can equally well be used to evaluate the probability of a DAG, $\mathfrak{D}$, of interest by instead accumulating the log probability of the sequence at each distribution encountered during the execution of the program. As such it defines a distribution over DAG serializations.

Algorithm D.1: *Pseudocode for* DOG-GEN

Input:

- Action networks for node addition, building block molecular identity, and connectivity choice: $\mathsf{na}(\cdot)$, $\mathsf{bbmi}(\cdot)$, $\mathsf{cc}(\cdot)$;
- Reaction predictor, $\mathsf{Product}(\cdot)$;
- Context RNN, $\boldsymbol{c}_j = \mathsf{GRU}(\boldsymbol{c}_{j-1}, \boldsymbol{e}_j)$;
- Continuous latent variable, $\boldsymbol{z}$;
- Linear projection for mapping continuous latent variable to RNN's initial hidden vector, $\mathsf{Lin}(\cdot)$;
- Graph neural network, $\mathsf{GNN}(\cdot)$, for computing molecule embeddings;
- Set of available (e.g. cheap, easy-to-obtain, etc.) building blocks, $\mathcal{B}$, to use as possible reactants;
- Learnable embeddings for abstract actions: $\boldsymbol{h}_{\text{'B'}}$, $\boldsymbol{h}_{\text{'P'}}$, $\boldsymbol{h}_{\not\rightarrow_{\mathcal{I}}}$, and $\boldsymbol{h}_{\not\rightarrow_{\mathcal{F}}}$.

Result:

- Complete molecule synthesis DAG, $\mathfrak{D}$;
- Final product molecule, $\mathcal{M}_I$.

Initialize DAG $\mathfrak{D} \leftarrow [V_1 = \text{'B'}]$, Initialize $\mathsf{T} \leftarrow [\mathsf{T}_1 = \mathsf{A1}, \mathsf{T}_2 = \mathsf{A2}]$
Initialize molecule set $M \leftarrow \{\}$ and unused reactants set $U \leftarrow \{\}$ ▷ Track all molecules & unused molecules.
$\boldsymbol{c}_1 \leftarrow \mathsf{Lin}(\boldsymbol{z})$ ▷ $\boldsymbol{z}$ initializes the first hidden state of the RNN.
$\boldsymbol{e}_2 \leftarrow \boldsymbol{h}_{\text{'B'}}$ ▷ Initial input into RNN reflects that new node added on first step.
while $V_{|\mathfrak{D}|} \neq \not\rightarrow_{\mathcal{F}}$ **do** ▷ Loop until stop symbol.
  $j \leftarrow |\mathfrak{D}| + 1$ ▷ Update action index for this iteration of the loop.
  $\boldsymbol{c}_j \leftarrow \mathsf{GRU}(\boldsymbol{c}_{j-1}, \boldsymbol{e}_j)$ ▷ Update context.
  if $\mathsf{T}_j = \mathsf{A1}$ **then** ▷ Node addition.
    $\boldsymbol{w} \leftarrow \mathsf{na}(\boldsymbol{c}_j)$; $\boldsymbol{B} \leftarrow \text{STACK}([\boldsymbol{h}_{\text{'B'}}, \boldsymbol{h}_{\text{'P'}}])$
    logits $\leftarrow \boldsymbol{w}\boldsymbol{B}^t$ ▷ Form logits from dot products ($\cdot^t$ is transpose).
    $V_j \sim \text{softmax}(\text{logits})$ ▷ Sample (from categorical) new node type: (a) building block or (b) product.
    if $V_j = \text{'B'}$ **then** ▷ If new building block:
      $\mathsf{T}_{j+1} \leftarrow \mathsf{A2}$; $\boldsymbol{e}_{j+1} \leftarrow \boldsymbol{h}_{\text{'B'}}$ ▷ next time pick its identity.
    else if $V_j = \text{'P'}$ **then** ▷ If new product:
      $\mathsf{T}_{j+1} \leftarrow \mathsf{A3}$; $\boldsymbol{e}_{j+1} \leftarrow \boldsymbol{h}_{\text{'P'}}$ ▷ next time connect it up.
      Initialize intermediate reactant set $R \leftarrow \{\}$ ▷ Will temporarily store "working" reactants.
      stop_actions $\leftarrow [\boldsymbol{h}_{\not\rightarrow_{\mathcal{F}}}]$ ▷ You cannot stop for intermediate product until at least one reactant.
  else if $\mathsf{T}_j = \mathsf{A2}$ **then** ▷ Building block molecular identity.
    $\boldsymbol{w} \leftarrow \mathsf{bbmi}(\boldsymbol{c}_j)$; $\boldsymbol{B} \leftarrow \text{STACK}([\mathsf{GNN}(g) \text{ for } g \text{ in } \mathcal{B} \setminus M])$
    logits $\leftarrow \boldsymbol{w}\boldsymbol{B}^t$
    $V_j \sim \text{softmax}(\text{logits})$ ▷ Sample (from categorical) building block molecule.
    $\mathsf{T}_{j+1} \leftarrow \mathsf{A1}$; $\boldsymbol{e}_{j+1} \leftarrow \mathsf{GNN}(V_j)$
    $M \leftarrow M \cup \{V_j\}$; $U \leftarrow U \cup \{V_j\}$
  else if $\mathsf{T}_j = \mathsf{A3}$ **then** ▷ Connectivity choice.
    $\boldsymbol{w} \leftarrow \mathsf{cc}(\boldsymbol{c}_j)$; $\boldsymbol{B} \leftarrow \text{STACK}([\mathsf{GNN}(g) \text{ for } g \text{ in } M \setminus R] + \text{stop\_actions})$
    logits $\leftarrow \boldsymbol{w}\boldsymbol{B}^t$
    $V_j \sim \text{softmax}(\text{logits})$ ▷ Sample from categorical: (a) molecule to connect to, or (b) to stop.
    if $V_j = \not\rightarrow_{\mathcal{I}}$ **then** ▷ Selected an intermediate product so run the reaction.
      $\mathcal{M}^{\text{new}} \leftarrow \mathsf{Product}(R)$
      $M \leftarrow M \cup \{\mathcal{M}^{\text{new}}\}$; $U \leftarrow U \cup \{\mathcal{M}^{\text{new}}\}$
      $\mathsf{T}_{j+1} \leftarrow \mathsf{A1}$; $\boldsymbol{e}_{j+1} \leftarrow \boldsymbol{h}_{\not\rightarrow_{\mathcal{I}}}$ ▷ Finished product so make new node again next time.
    else if $V_j \in M$ **then** ▷ Selected an extra reactant.
      $R \leftarrow R \cup \{V_j\}$ ▷ Update reactant set.
      $U \leftarrow U \setminus \{V_j\}$ ▷ Remove from pool of "unused" molecules.
      $\mathsf{T}_{j+1} \leftarrow \mathsf{A3}$; $\boldsymbol{e}_{j+1} \leftarrow \mathsf{GNN}(V_j)$ ▷ Continue to add edges next time around.
      stop_actions $\leftarrow [\boldsymbol{h}_{\not\rightarrow_{\mathcal{I}}}, \boldsymbol{h}_{\not\rightarrow_{\mathcal{F}}}]$ ▷ Now you can stop for both final or intermediate product.
  Update $\mathfrak{D} \leftarrow [V_1, \ldots, V_j]$; $\mathsf{T} \leftarrow [\mathsf{T}_1, \ldots, \mathsf{T}_j]$
Predict final product $\mathcal{M}_I \leftarrow \mathsf{Product}(R \cup U)$ ▷ The final product considers both R and U.
return $\mathfrak{D}$, $\mathcal{M}_I$

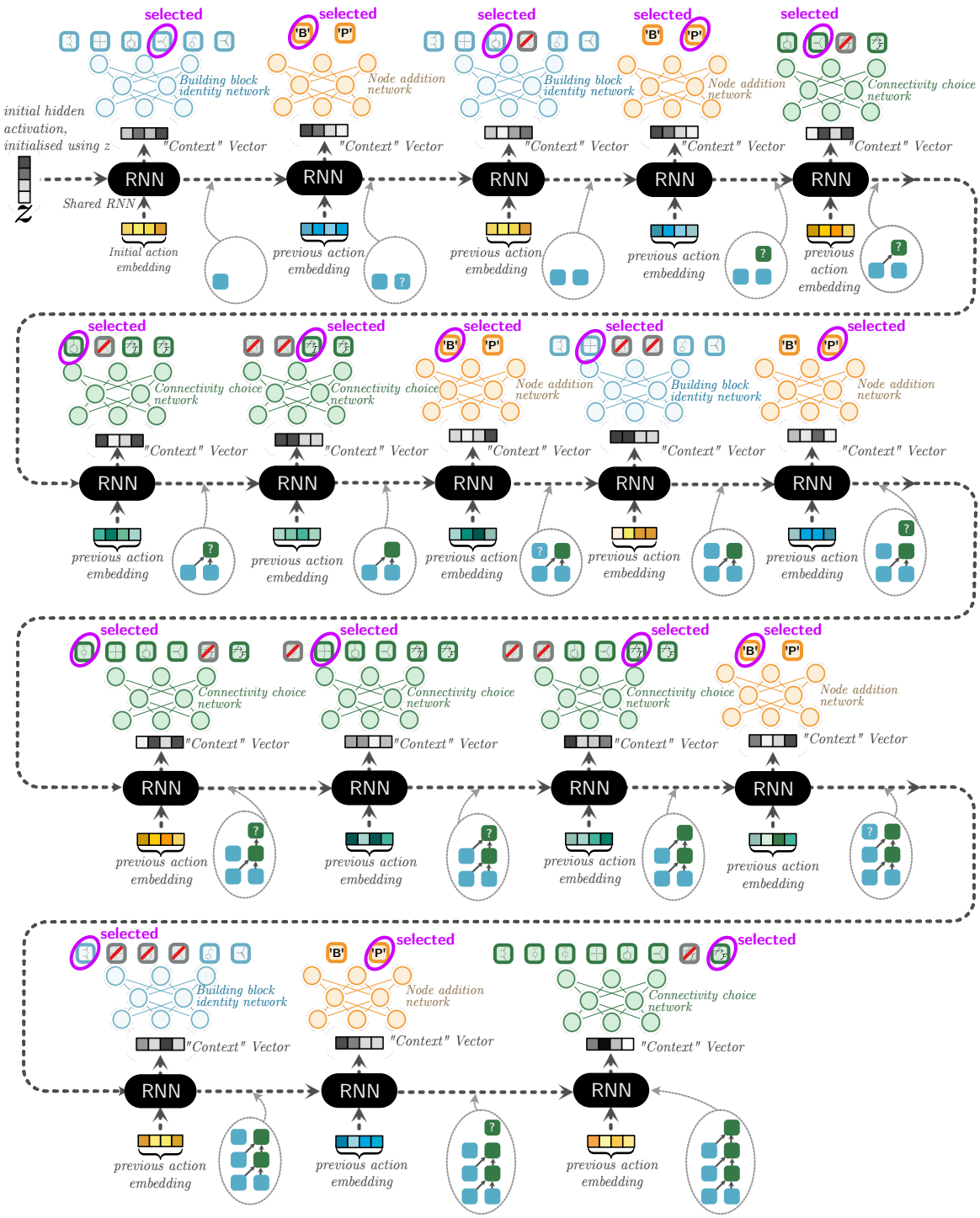


Figure D.2 *This is an expanded version of Figure 5.4 in the main text showing all the actions required to produce the example DAG for paracetamol (see also Figures 5.3 and 5.1 in Chapter 5). A shared RNN (recurrent neural network) provides a context vector for the different action networks. Based on this context vector, each type of action network chooses an action to take (some actions are masked out as they are not allowed, for instance suggesting a building block already in the graph, or choosing to make an intermediate product before choosing at least one reactant). Note that the embeddings of the molecular graphs are computed using a GNN (graph neural network; see Section 2.2.2). The initial hidden vector of the shared RNN is initialized using a latent vector* $\mathbf{z}$ *in our autoencoder model* DoG-AE*; in* DoG-Gen *it is set to a constant. The state of the DAG at each stage of the generative process is indicated in the dotted gray circles.*

D.3 Synthesizability score

The synthesizability score is defined as the geometric mean of the nearest neighbor reaction similarities:

$$\sqrt[|R|]{\prod_{r \in R} \kappa\big(r, \mathrm{nn}(r)\big)}, \tag{D.1}$$

where R is the list of reactions making up a synthesis DAG, $r \in R$ are the individual reactions in the DAG, $\mathrm{nn}(r)$ is the nearest neighbor reaction in the chemical literature in Morgan fingerprint space, and $\kappa(\cdot,\cdot)$ is the Tanimoto similarity, defined using Morgan reaction fingerprints (Schneider et al., 2015a).

D.4 Experimental details

This section reports more specific details about the architectures of the models used in the experiments (further details can also be found in our code, `https://github.com/john-bradshaw/synthesis-dags`). This section follows a similar format to the equivalent section in the previous appendix, first describing the hyperparameters used for the models we proposed, before providing a summary of the implementations used for the baselines. The section ends with a discussion on the use of the fine-tuning approach in the optimization experiments.

D.4.1 Architecture and training details

Here we describe in turn the architecture and training details for our DoG-Gen and DoG-AE models.

DoG-Gen (as an autoregressive model)

Forming molecule embeddings For forming molecule embeddings we used a GGNN (Gated Graph Sequence Neural Network; § A.1). This network operated on the atom features described in Table D.1. The GGNN was run for 5 rounds of message passing (i.e. $L = 5$) to form 80 dimensional node embeddings; these node embeddings were aggregated into a 160 dimensional molecule embedding through a linear projection and weighted sum.

RNN For generating the context vector we used a 3 layer GRU RNN (gated recurrent unit recurrent neural network) with 512 dimensional hidden layers. The action networks

Table D.1 *Atom features we use as input to the GGNN. These are calculated using RDKit.*

Feature	Description
Atom type	One hot (72 possible elements in total)
Atomic number	Integer
Acceptor	Boolean (accepts electrons)
Donor	Boolean (donates electrons)
Hybridization	One hot (SP, SP2, SP3)
Part of an aromatic ring	Boolean
H count	Integer

used were feedforward neural networks with one hidden layer of dimension 28 and ReLU (rectified linear unit) activation functions. For the RNN, the initial hidden vector was set using a learned constant (i.e. this vector was initialized the same way each time, unlike DoG-AE).

Training We trained our model for 30 epochs, as we found (using our validation dataset) that training for longer would lead to overfitting.

Fine-tuning For property optimization (i.e. via the fine-tuning approach described in Algorithm 5.1), we started by evaluating the score on every synthesis DAG in our training and validation datasets; we then ran 30 stages of fine-tuning, sampling 7,000 synthesis DAGs at each stage and updating the weights of our model by using the best 1,500 DAGs seen at that point as a fine-tuning dataset. For both our model and the baselines, when reporting the GuacaMol benchmark property score, we reported the score obtained by the best individual molecule in the group.

We used Jug (Coelho, 2017), a parallelization framework, to parallelize over the GuacaMol optimization tasks.

Reaction predictor As described in the main text, we used the Molecular Transformer (Schwaller et al., 2019) for reaction prediction. We used pre-trained weights; these were obtained by training on a processed USPTO (Jin et al., 2017; Lowe, 2012) dataset without reagents. We treated the Transformer as a black box oracle and so made no further adjustments to its weights when training the rest of our model later.

DoG-AE

Forming molecule embeddings The way we formed molecule embeddings for DoG-AE was similar to the approach we used for DoG-Gen, discussed above. We again used a GGNN, this time for 4 propagation steps, and projected the final node embeddings down to a 50 dimensional space by using a learned linear projection. The node embeddings were then combined to form molecule embeddings through a weighted sum. The same GNN architecture was shared between the encoder and the decoder.

Encoder The encoder consisted of two GGNNs. The first, described above, created molecule embeddings which were then used to initialize the node embeddings in the synthesis DAG. The synthesis DAG node embeddings, which were 50 dimensional, were further updated using a second GGNN. This second GGNN performed 7 stages of message passing (i.e. $L = 7$), and the messages were passed forward on the DAG from the "leaf" nodes to the final product node. Finally, the node embedding of the final product molecule node in the DAG was passed through an additional linear projection in order to parameterize the mean and log variance of independent Gaussian distributions over each dimension of the latent variable.

Decoder For the decoder (i.e. the embedded DoG-Gen model) we used a 3 layer GRU RNN (Cho et al., 2014a) to compute the context vector. The hidden layers had a dimension of 200 and while training we used a dropout rate of 0.1. For initializing the hidden layers of the RNN we used a linear projection of $\boldsymbol{z}$ (the parameters of which we learned). The action networks were feedforward neural networks with one hidden layer (dimension 28) and ReLU activation functions. For the abstract actions (such as 'B'or 'P') we learned 50 dimensional embeddings, such that these embeddings had the same dimensionality as the molecule embeddings we computed.

Training We trained DoG-AE, with a 25 dimensional latent space, using the Adam optimizer (Kingma and Ba, 2014), an initial learning rate of 0.001, and a batch size of 64. We used an inverse multiquadratics kernel for computing the MMD term in the WAE objective.

DoG-AE was trained using teacher forcing for 400 epochs (each epoch took approximately 7 minutes) and we multiplied the learning rate by a factor of 0.1 after 300 and 350 epochs. DoG-AE obtained a reconstruction accuracy (on our held out test set) of 65% when greedily decoding (greedy in the sense of picking the most probable action at each stage of decoding). If we tried instead decoding by sampling 100 times from our model

and then sorting based on probability, we obtained a slightly improved reconstruction accuracy of 66%.

D.4.2 Baselines

The detail of the baselines used in Table 5.1 are shown below:

CGVAE (Liu et al., 2018): We used the official implementation:
`https://github.com/microsoft/constrained-graph-variational-autoencoder`
We decided to include element-charge-valence triplets that occur at least 10 times over all of the molecules in the training data. At generation time we pick one starting node at random.

SMILES LSTM (Segler et al., 2018a): We used the implementation available at:
`https://github.com/BenevolentAI/guacamol_baselines`

JT-VAE (Jin et al., 2018): We used the official implementation available at:
`https://github.com/wengong-jin/icml18-jtnn`
We used the updated version of this code, i.e. the `fast_jtnn` version.

GVAE (Kusner et al., 2017): We used our own re-implementation. We used a 72 dimensional latent space. We multiplied the KL term in the VAE loss by a parameter β (see Box 4.2); this β term was then gradually annealed in during training until it reached a final value of 0.3. We used a 3 layer GRU RNN (Cho et al., 2014a) for the decoder with 384 dimensional hidden layers. The encoder was a 3 layer bidirectional GRU RNN also with 384 dimensional hidden layers.

CVAE (Gómez-Bombarelli et al., 2018): We used our own implementation, using the same hyperparameters as for our implementation of the GVAE described above.

GraphVAE (Simonovsky and Komodakis, 2018): We used our own re-implementation. When training the GraphVAE on our datasets we excluded any molecules with greater than 20 heavy atoms, as this procedure was found in the original paper to give better performance when training on ZINC (Simonovsky and Komodakis, 2018, § 4.3). We used a 40 dimensional latent space, a GGNN (§ 2.2.2) for the encoder, and used the max-pooling graph matching procedure (see Simonovsky and Komodakis, 2018, § 3.4) during training.

For the CVAE, GVAE, and GraphVAE baselines, we used our own implementations, as noted above. We tuned the hyperparameters of these models on the ZINC or QM9 datasets

so that we were able to get at least similar (and often better) results compared to those originally reported in Kusner et al. (2017); Simonovsky and Komodakis (2018).

D.4.3 Rationale behind using the fine-tuning approach for optimization

In this subsection we provide a short commentary on why we ended up using the fine-tuning technique (Alg. 5.1) for the optimization experiments in Chapter 5.

For the optimization experiments in Chapter 5, we moved away from the local optimization approach used in Chapter 4 (in particular see Section 4.3.2). This is because we found such an approach less effective for DoG-AE. In particular, we found it difficult to train an autoencoder for molecular synthesis DAGs for which the latent space was smooth in terms of the final molecule's property score. To be specific, by moving a short distance in latent space, we might change one building block in our DAG and end up with a final molecule with very different properties, making optimization difficult (this phenomenon was also seen with Molecule Chef but to a much lesser extent – see Section 4.3.2). In contrast, we found adapting the fine-tuning technique from Segler et al. (2018a) empirically worked well with little setup.

We believe that discovering the best way to utilize generative models of molecules for molecular optimization is still an open problem. In our experience, autoencoder-type models, such as Molecule Chef or DoG-AE, generally struggle with the more challenging optimization tasks of Chapter 5. Often to perform well on these tasks, we need models to extrapolate and generate very different looking molecules, an area which autoencoders are perhaps less well suited for. There has been some recent work looking to address this using retraining techniques (Tripp et al., 2020), which are not too dissimilar in spirit to the fine-tuning approach we used[2]. However, a challenge with fine-tuning approaches, and indeed many current optimization techniques, is that of sample efficiency. We think this is an interesting direction to explore in the future and discussed some ideas in this vein in Section 5.5.2.

[2]Note, however, that Tripp et al. (2020)'s method also has some key differences to the fine-tuning approach we used, such as the continuous weighting of *all* datapoints during retraining/fine-tuning as well as the use of a second optimization routine.

D.5 Additional experiments

D.5.1 Further results from the GuacaMol optimization tasks

This section provides more results from the GuacaMol optimization tasks described in the main text (§ 5.4.2). Similar to the analysis in the main text, these have been produced by considering the top molecules returned by each method for each task (note that we considered only the top molecules each method returned, and we limited the number to a maximum of 100).

Figure D.3 shows the scores of the best molecules found on each of the tasks by the various methods, particularly distinguishing between the cases in which (i) no synthetic route can be found, (ii) a synthetic route can be found, and (iii) a synthetic route can be found *and* the found route has a synthesizability score (§ D.3) over 0.9.

Tables D.2, D.3, and D.4 provide a more detailed breakdown of the results presented in Table 5.2 in the main text. Table D.2 shows (for each task) the fraction of the top 100 molecules proposed by each method for which a synthetic route can be found. Table D.3 shows the average synthesis score over the top 100 molecules proposed by each method for each task. Table D.4 shows (for each task) the median number of steps required to synthesize the top 100 molecules found by each method; here, the median has been calculated considering only those molecules for which a synthetic route can be found.

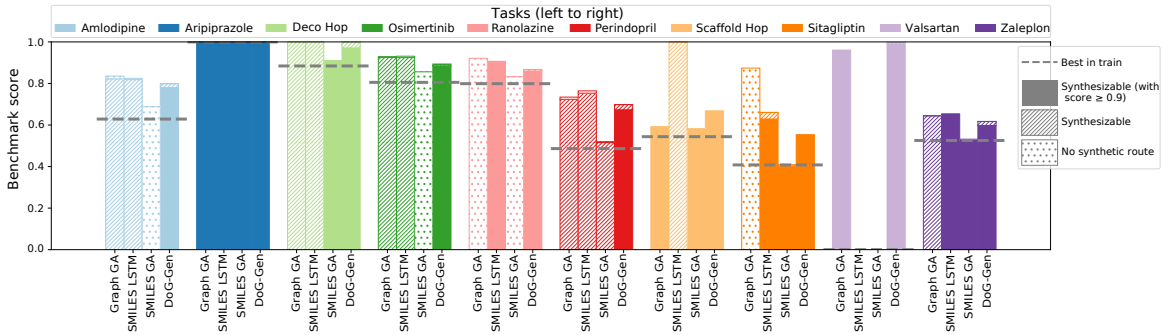


Figure D.3 *The score of the best molecule found by the different approaches over a series of ten GuacaMol tasks (Brown et al., 2019, § 3.2). Scores range between 0 and 1, with 1 being the best. We also differentiate between the synthesizability of the molecules found (judged using a computer-aided synthesis planning tool). This is shown in the figure through the use of different hatching styles on the bars: (i) the circular hatched bar shows the score of the best molecule found (regardless of synthesizability), (ii) the diagonally hatched bar shows the score of the best molecule found with a synthetic route, and (iii) the solid bar shows the score of the best molecule found with a synthetic route and a synthesizability score (§D.3) greater than or equal to 0.9. Note that later bars occlude earlier bars. The dotted gray line for each task represents the score one obtains if picking the best molecule in the training set.*

Table D.2 *Fraction of the top 100 molecules suggested by each method for which a synthetic route can be found.*

	Graph GA	SMILES LSTM	SMILES GA	DoG-Gen
Amlodipine	0.52	0.13	0.00	1.00
Aripiprazole	0.98	0.72	0.29	0.99
Deco Hop	0.09	0.05	0.16	0.95
Osimertinib	0.27	0.09	0.00	0.67
Perindopril	0.25	0.15	0.02	0.82
Ranolazine	0.00	0.77	0.00	0.63
Scaffold Hop	0.54	0.02	0.68	0.91
Sitagliptin	0.00	0.96	0.50	0.98
Valsartan	0.85	0.87	0.89	1.00
Zaleplon	0.72	1.00	0.53	1.00

Table D.3 *Average synthesizability score score (see § D.3) of the top 100 molecules suggested by each method.*

	Graph GA	SMILES LSTM	SMILES GA	DoG-Gen
Amlodipine	0.35	0.09	0.00	0.86
Aripiprazole	0.91	0.62	0.24	0.98
Deco Hop	0.07	0.04	0.12	0.74
Osimertinib	0.20	0.06	0.00	0.57
Perindopril	0.19	0.09	0.02	0.68
Ranolazine	0.00	0.69	0.00	0.55
Scaffold Hop	0.36	0.02	0.52	0.71
Sitagliptin	0.00	0.62	0.42	0.83
Valsartan	0.72	0.79	0.82	0.88
Zaleplon	0.48	0.87	0.48	0.82

Table D.4 *Table showing the median number of synthetic steps for each molecule found in the top 100 (median calculated only over synthesizable molecules). Hyphens ("-") indicate no synthesizable molecule was suggested in the top 100 by that method on that task.*

	Graph GA	SMILES LSTM	SMILES GA	DoG-Gen
Amlodipine	7.0	6.0	-	3.5
Aripiprazole	3.0	6.0	5.0	2.0
Deco Hop	5.0	8.0	3.5	5.0
Osimertinib	8.0	9.0	-	7.0
Perindopril	7.0	11.0	6.0	7.0
Ranolazine	-	7.0	-	8.0
Scaffold Hop	7.0	8.0	5.0	4.0
Sitagliptin	-	7.0	3.0	3.0
Valsartan	6.0	3.0	2.0	3.0
Zaleplon	6.0	3.0	4.0	3.0

D.5.2 Retrosynthesis

Our generative model of synthesis DAGs, DoG-Gen, can be seen as a parameterizable mapping from a vector of real numbers to a synthesis DAG. As a module it can be mixed and matched in different ML frameworks, as we have already seen in the main thesis with, for instance, DoG-AE. In this section we describe some preliminary results with a third model architecture, called RetroDoG. This model consists of the composition of a GNN with the DoG-Gen model to produce a learnable mapping from a molecular graph to a synthesis DAG. By training this model on pairs of product molecules and their associated synthesis DAGs, we can use this model to perform retrosynthesis (i.e. predict how a particular product can be made). Much like the approach described in Section 4.3.3 for Molecule Chef, the model described in this section can additionally allow one to feed in a potentially hard- or impossible-to-synthesize molecule and obtain a similar molecule that is *easy to synthesize*, which is impossible to do with current planners.

In order to qualitatively assess such a model, we train RetroDoG on the same DAG dataset described in Section D.1. At test time, we sample 200 DAGs from our model and then sort them based on the Tanimoto similarity (using Morgan fingerprints) between the final product molecule of the DAG and the original molecule fed into RetroDoG, before picking the best one (note that this uses only the input data and not the true target data). In Figures D.4 and D.5 we show the result of this on three molecules taken from the WHO Essential Medicines List (World Health Organization, 2019). Although we do not find a route to the exact molecule of interest, we often decode to similar final product molecules.

In particular, we find that RetroDoG finds molecules that are often as similar (measured again using the Tanimoto similarity between the fingerprints) as the closest molecule to the target in our original training dataset.

We should point out, much like the work in Section 4.3.3, that we do not expect this preliminary approach to currently be competitive with complex synthesis planning tools such as the one proposed by Segler et al. (2018b). These complex tools work in a top-down manner assessing many different routes. RetroDoG in contrast tries to construct the DAG in a bottom-up manner and is not able to roll back and adjust already made choices based on new data. Having said that, we believe such an approach as RetroDoG, or the method described in Section 4.3.3, may be worthy of future research interest. For instance, it could be useful in combination with more complex tools to amortize the cost of searching for synthetic routes (see the discussion in Section 6.2.2).

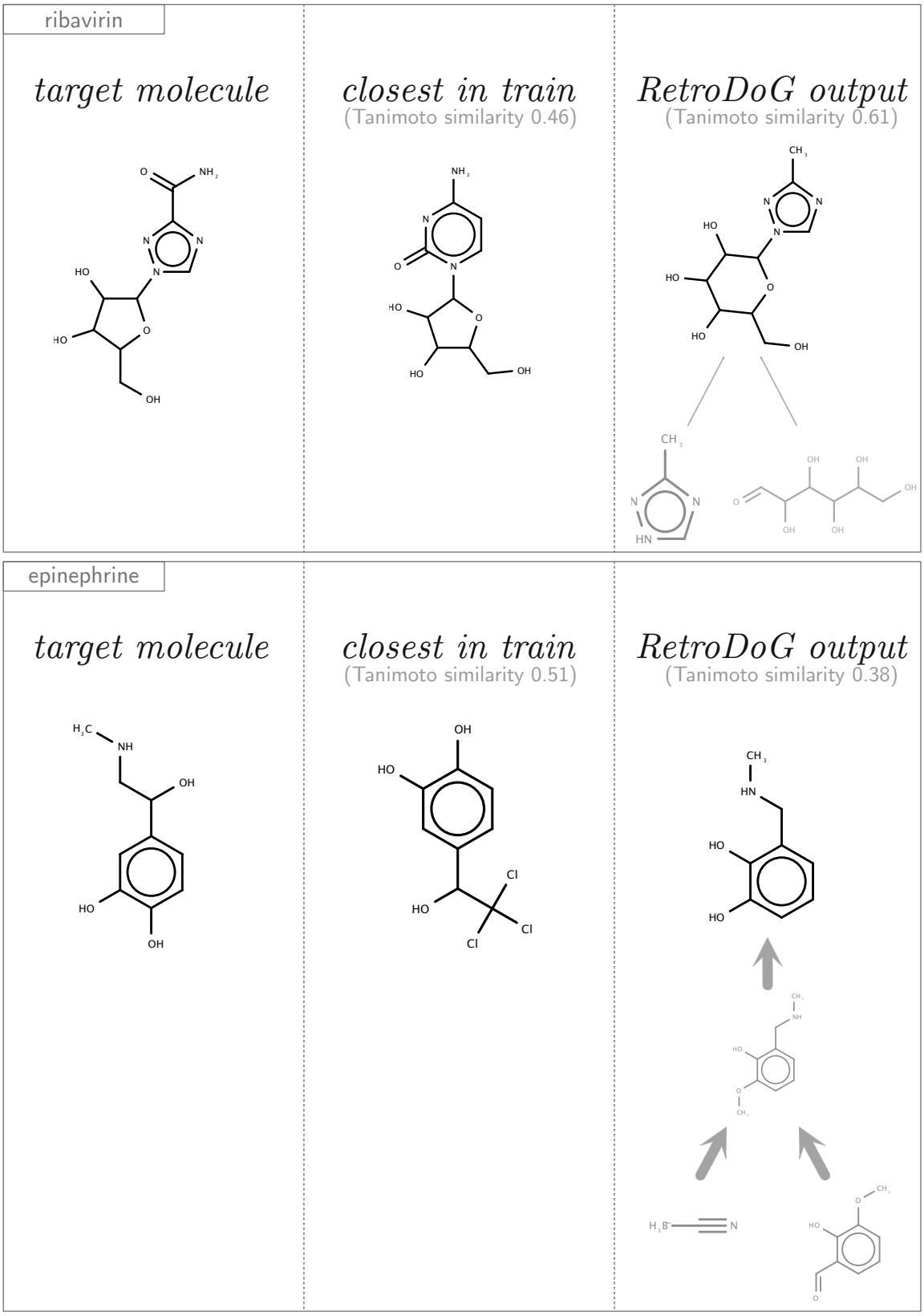


Figure D.4 *RetroDoG suggested DAGs for ribavirin and epinephrine.*

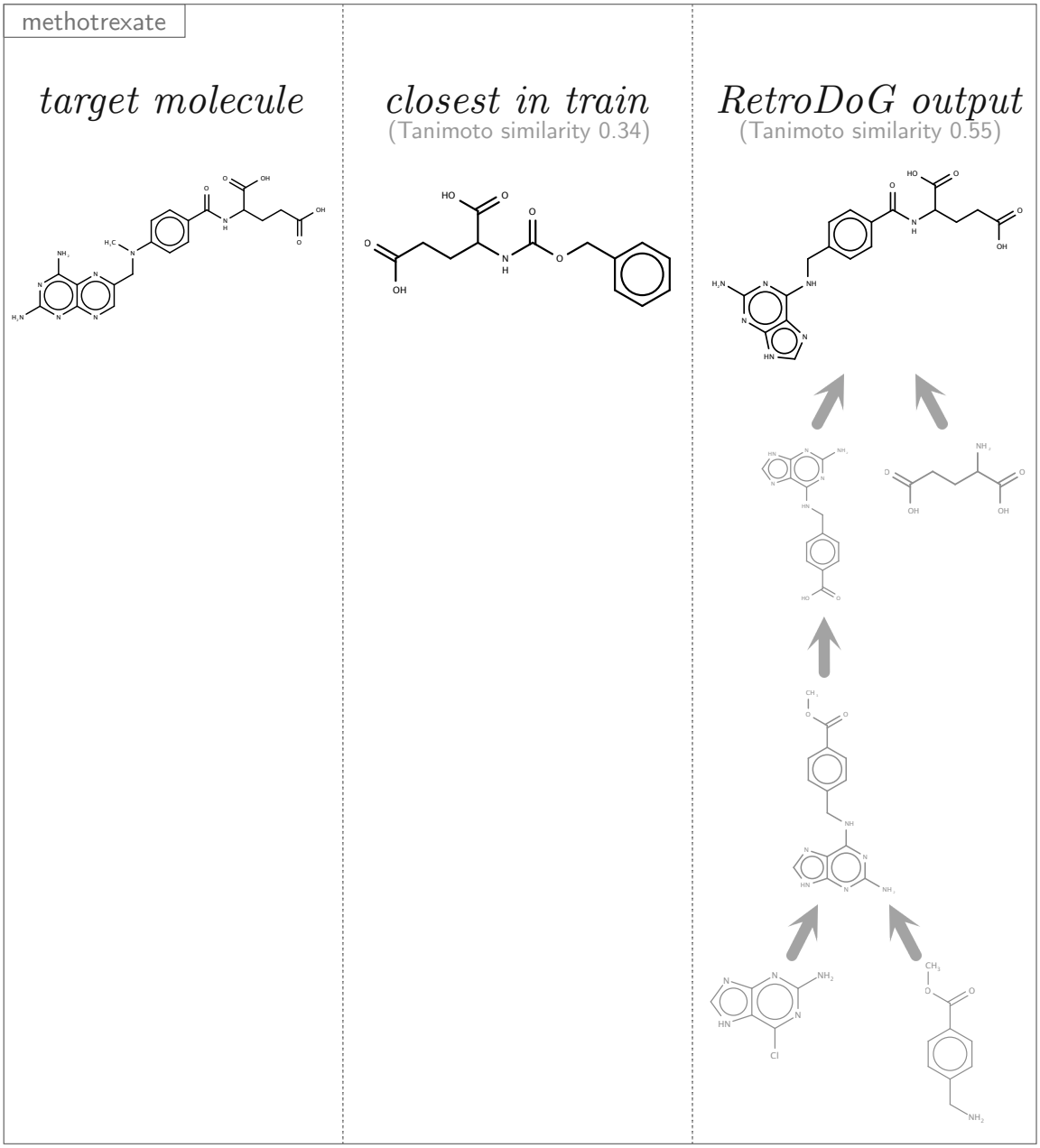


Figure D.5 *RetroDoG suggested DAG for methotrexate.*

D.5.3 Comparison to the SYNOPSIS algorithm

In this section we compare to a re-implemented version of the SYNOPSIS algorithm as an additional baseline (Vinkers et al., 2003). This method represents one of the first reaction-driven de novo design algorithms (see Box 4.1) and uses a simulated annealing–based optimization approach. The algorithm works as follows (generating ≈ 100,000 molecules per task):

1. A pool of molecules is initialized using all of the available building blocks and training molecules provided to the other models.

2. Three molecules are sampled from the pool according to a Boltzmann distribution: $p(\mathcal{M}_i) \propto \exp\left(-(\mathsf{obj}^{\max} - \mathsf{obj}(\mathcal{M}_i))/T_c\right)$, where $\mathcal{M}_i$ is the molecule being considered, $\mathsf{obj}(\cdot)$ the objective function, $\mathsf{obj}^{\max}$ the maximum seen objective value, and T_c the current "temperature".

3. A reaction scheme for these molecules is randomly sampled (we use the schemes published in Button et al., 2019; Chevillard and Kolb, 2015) and the reactions carried out. If the reaction requires a reaction partner, up to 64 suitable building blocks are selected (to use as reactants) by sampling uniformly from all building blocks applicable under the reaction scheme.

4. The resulting molecules are scored and then added to both an output collection (eventually returned) and the initial pool. The procedure then returns to step 2 and steps 2-4 are repeated until 1,000 iterations of the resulting loop have been completed.

Table D.5 shows the results of running this re-implementation of Vinkers et al. (2003)'s algorithm. The results show that DoG-Gen performs well in comparison to this older reaction-driven de novo design algorithm.

Table D.5 *Score of the best molecule found by* DoG-Gen *and the SYNOPSIS-inspired (Vinkers et al., 2003) discrete optimization algorithm on the GuacaMol tasks.*

	DoG-Gen	SYNOPSIS
Amlodipine	0.80	0.63
Aripiprazole	1.00	0.87
Deco Hop	1.00	0.88
Osimertinib	0.89	0.84
Perindopril	0.70	0.55
Ranolazine	0.87	0.83
Scaffold Hop	0.67	0.54
Sitagliptin	0.55	0.43
Valsartan	1.00	0.00
Zaleplon	0.62	0.52