

Molecular Optimization using Deep Learning

Extensions of the Transformer for Molecular Optimization

Master's thesis in Computer science and engineering

Marcus Forsberg
Felix Mattsson

MASTER'S THESIS 2021

Molecular Optimization using Deep Learning

Extensions of the Transformer for Molecular Optimization

Marcus Forsberg
Felix Mattsson



UNIVERSITY OF
GOTHENBURG



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Molecular Optimization using Deep Learning
Extensions of the Transformer for Molecular Optimization
Marcus Forsberg
Felix Mattsson

© Marcus Forsberg, Felix Mattsson, 2021.

Supervisor: Yinan Yu, Department of Computer Science and Engineering
Advisor: Jiazhen He, AstraZeneca
Examiner: Alexander Schliep, Department of Data Science and AI

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Molecular Optimization of left molecule (CHEMBL3953242), resulting in the generated molecules displayed to the right (CHEMBL3896788, CHEMBL3936463)

Typeset in L^AT_EX
Gothenburg, Sweden 2021

Molecular Optimization using Deep Learning
Extensions of the Transformer for Molecular Optimization
Marcus Forsberg
Felix Mattsson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Over the recent years, the development in deep learning has provided new approaches to molecular optimization. Molecular optimization aims to find structurally similar molecules to a given starting molecule, yielding specified improvements in terms of different molecular properties. By representing molecules as SMILES, an approach to encode molecules as strings of tokens, molecular optimization can be framed as a machine translation problem, where starting molecules are *translated* to molecules with optimized properties. Previous work has shown success for the Transformer known from natural language processing [1, 2] in the area of molecular optimization. The thesis covers two extensions of the developed Transformer model in [1] through curriculum learning and Core-Fixed formulation. Through curriculum learning, training is structured through a sequence of tasks (curriculum) based on increasing difficulty. The curriculum could either be determined while training a model (machine-based) or manually (human heuristic-based). The thesis explores various approaches to human-based curriculum learning. For the other extension, Core-Fixed formulation, the thesis provides an approach to reformulating the input and output of the original model [1], which involves specifying in the input to the translation model which part that should be fixed (core) and which part that should be exchanged (R-group) to optimize the complete molecule's properties. The results show advantages both in training time and molecule generation performance using the Core-Fixed formulation. For curriculum learning, the results do not indicate a clear improvement. The thesis suggests looking into more sophisticated curriculum learning approaches.

Keywords: Molecular Optimization, Matched Molecular Pairs, Transformer, ADMET, Master's Thesis.

Acknowledgements

First and foremost, we would like to thank our supervisor at AstraZeneca, Jiazhen He, who both created the work for which this entire thesis is based, and who has helped us through out the project with useful insights and feedback. Without her work this thesis would not be possible. Additionally, we thank the entire Molecular AI team of AstraZeneca for their help with setting up the computer environment and preparing the code for which our implementations are based. In particular, we would like to mention Vendy Fialkova, Graduate Scientist at AstraZeneca, for her feedback on the report and Ola Engkvist, the director of the Molecular AI team, for his continuous inputs that have helped to shape the project.

We would also like to thank our academic supervisor at Chalmers, Yinan Yu, and our opponent Felix Nordén, for their constructive and useful feedback on the report. Finally, we thank our families and friends for their continuous support.

Marcus Forsberg, Felix Mattsson, Gothenburg, February 2021

Abbreviations

The following list shows frequent abbreviations used in this report.

- **SMILES** - **S**implified **M**olecular **I**nput **L**ine **E**ntry **S**ystem
- **MMP** - **M**atched **M**olecular **P**air (molecules that differ only by a single transformation)
- **NLP** - **N**atural **L**anguage **P**rocessing
- **AZ** - **A**stra**Z**eneca
- **ADMET** - **A**bsorption, **D**istribution, **M**etabolism, **E**xcretion and **T**oxicity
- **CL** - **C**urriculum **L**earning
- **CF** - **C**ore-**F**ixed



Contents

List of Figures	xv
List of Tables	xvii
1 Introduction	1
1.1 Drug Discovery	1
1.2 Challenges in Drug Discovery	1
1.3 String Based Molecular Representation	2
1.4 Project Description: Extensions of Previous Model	3
1.4.1 Curriculum Learning	3
1.4.2 Core-Fixed Molecular Optimization	4
1.5 Thesis Outline	4
2 Theory	7
2.1 Transformer	7
2.1.1 Token Embedding	7
2.1.2 Positional Encoding	9
2.1.3 Attention	9
2.1.4 Encoder	10
2.1.5 Decoder	10
2.1.6 Sequence Generation	11
2.2 Transformer for Molecular Optimization	12
2.2.1 Considered Molecular Properties	12
2.3 Model Description	12
2.4 Curriculum Learning	14
2.4.1 Difficulty Assessment	15
2.4.2 Curriculum Arrangement	15
3 Methods	17
3.1 Platform	17
3.2 Data Preparation	17
3.2.1 Preparation of Matched Molecular Pairs	17
3.2.2 Property Representation	18
3.3 Curriculum Learning	19
3.3.1 Simulation of Missing Property Values	19
3.3.2 Difficulty Assessment	20
3.3.3 Curriculum Arrangement	21

3.3.4	Baseline	22
3.3.5	Training and Validation	23
3.3.6	Test Sets	23
3.3.7	Evaluation Metrics	24
3.4	Core-Fixed Molecular Optimization	25
3.4.1	Input and Output Representations	25
3.4.2	Baselines	25
3.4.3	Training and Validation	25
3.4.4	Test Sets	27
3.4.5	Evaluation Metrics	28
4	Results	29
4.1	Curriculum Learning	29
4.1.1	Comparison of Exploratory Difficulty Assessments	29
4.1.1.1	Training and Validation	29
4.1.1.2	Sample-Based Molecule Generation	31
4.1.2	Comparison of Property-Based with Comparative Difficulty Assessments	32
4.1.2.1	Training and validation	32
4.1.2.2	Sample-Based Molecule Generation	34
4.1.3	Computational Time	34
4.2	Core-Fixed Molecular Optimization	35
4.2.1	Training and Validation	35
4.2.2	Deterministic Molecule Generation	36
4.2.3	Sample-Based Molecule Generation	37
4.2.4	Novel R-Group Samples	40
4.2.5	Example of Baseline’s Inability to Keep the Core	40
4.2.6	Computational Time	41
5	Discussion	43
5.1	Curriculum learning	43
5.1.1	Training and Validation	43
5.1.2	Molecule Generation	44
5.1.3	Future Work	44
5.2	Core-Fixed Molecular Optimization	45
5.2.1	Training and Validation	45
5.2.2	Molecule Generation	45
5.2.3	Future Work	46
6	Conclusion	47
	Bibliography	49
A	Additional Results for Curriculum Learning	I
A.1	Choice of Epoch Numbers for Difficulty Assessments	I
A.2	Additional Comparative Results	III

B	Additional Results for Core-Fixed Transformer	V
B.1	Required Number of Samples to Generate Molecules	V
B.2	Hyperparameter Tuning	V
C	Testing for Significant Difference in Distributions	VII

List of Figures

1.1	Comparison of machine translation and molecular optimization	2
1.2	The structural formula and SMILES-string for the molecule <i>caffeine</i> .	3
1.3	An example of matched molecular pair and corresponding properties.	4
2.1	The Transformer architecture introduced in [2]	8
2.2	An example of input and output of the previous model [1].	13
2.3	Visualization of difficulty based buckets in curriculum learning	16
3.1	All encoded property change tokens	18
3.2	Visualization showing how the data was masked to simulate practical scenario of missing properties.	20
3.3	Training data division for Property-, Length- and Token Rarity based difficulties with 4 buckets	22
3.4	Input and output chain of the Core-Fixed model	26
4.1	Training and validation loss over training time for the Exploratory difficulty assessments in curriculum learning	30
4.2	Validation accuracy over training time and smoothed validation accuracy over epochs for the Exploratory difficulty assessments in curriculum learning	30
4.3	Training and validation loss over training time for the Comparative difficulty assessments in curriculum learning	33
4.4	Validation accuracy over training time and, smoothed validation accuracy over epochs for the Comparative difficulty assessments in curriculum learning	33
4.5	Training hours required for each of the Exploratory and Comparative models, including the baseline.	35
4.6	Train and validation loss over epoch, and validation accuracy over epoch, for the Core-Fixed model	36
4.7	Number of generated molecules with desirable properties per source molecule when using the Core-Fixed model and its associated Baseline	39
4.8	Top 10 most frequent novel R-groups found by the Core-Fixed model	40
4.9	Example of a source molecule for which the baseline failed to keep the core	40
4.10	Generations based on the source molecule in Figure 4.9 for which the Baseline failed to keep the core	41

A.1	Validation loss and validation accuracy for the three difficulty assessments	II
A.2	<i>Comparative</i> validation loss and validation accuracy for the length and token based difficulty assessments	III
B.1	Distribution of number of samples required to generate 10 unique and valid molecules for the Core-Fixed model and its Baseline	V
B.2	Training loss, validation loss and validation accuracy over training epoch for the 8 hyperparameter combinations yielding the best performing Core-Fixed Transformer	VI

List of Tables

2.1	Hyperparameters for the Transformer used in [1]	14
3.1	The amount of molecules and molecular pairs still having its properties after the masking	20
4.1	Comparison of the generation performance of the Explorative difficulty assessments (DA) and the corresponding baseline	31
4.2	Comparison of the generation performance of the Comparative difficulty assessments and the corresponding baseline	34
4.3	Comparison of the Core-Fixed model and its Baseline, when using greedy decode with one generated molecule per starting molecule	36
4.4	Comparison of the performance of the Core-Fixed model and its corresponding baselines	37
4.5	Comparison of run-times for Core-Fixed model and corresponding baselines	41
B.1	The 8 hyperparameters that yielded the best performing Core-Fixed Transformer models	VI

1

Introduction

In this chapter we give a background of molecular optimization and the work that this thesis is based on. Furthermore, we briefly introduce the concept of representing molecules as SMILES strings in Section 1.3, and the main tasks that we consider in this thesis in Section 1.4.

1.1 Drug Discovery

The use of drugs as medication by humans have been around for thousands of years, with one early example being the use of herbal medicines, dating back to the Stone Age [3]. Ever since then, more and more drugs have been discovered, and as our society has grown and made significant advancements in science and technology, so have our methods for discovering new drugs. Historically, drugs have been discovered by identifying the active substances in already known remedies. More recently, the methods have been expanded by large-scale companies, through usage of chemical databases, clinical trials, advancements in biochemistry and much more. [4]

When designing a drug, one typically aims for a good balance of different molecular properties. Common considerations are molecule’s physiochemical properties, absorption, distribution, metabolism, elimination and toxicity (ADMET) properties. [1] Typically, a promising molecule needs to be improved to gain a balance between multiple properties – a problem known as *molecular optimization*. Traditionally, molecular optimization has relied on chemists utilizing their knowledge and intuition to apply chemical transformations to a promising molecule. Here the matched molecular pair (MMP) analysis [5, 6], which compares the properties of two molecules that differ only by a single chemical transformation, has been used extensively [7, 8, 9]. The chemist’s approach is based on the assumption that similar molecules possess similar properties, which has been useful to make approximations of molecules’ properties through history.

1.2 Challenges in Drug Discovery

Considering the drug-like space is estimated to contain $10^{23} - 10^{60}$ molecules [10], an exhaustive search for potential drugs is not preferred. Moreover, *only* 10^8 substances have ever been synthesized [11]. At AstraZeneca (AZ), developing suitable models to find potential drugs is of high relevance since it can help accelerating the

drug-development process and in the end save lives.

The recent development in deep learning has enabled further potential for finding molecules with desirable properties. Previous deep learning approaches have, however, often ignored the domain knowledge of chemical transformations. At AstraZeneca, a recent deep learning model has been developed, and is introduced in [1]. It is based on the chemical transformations (i.e. MMPs), which reflect the chemist’s intuition. The molecular optimization problem is framed as a machine translation problem in natural language processing (NLP), where a promising molecule, represented as a string, is translated into a similar molecule with optimized properties, much like how a sentence in English would be translated into a sentence in French, see Figure 1.1. More specifically, the work at AstraZeneca [1] showed potential for a model based on the *Transformer*, which is a neural network architecture used in NLP [2].

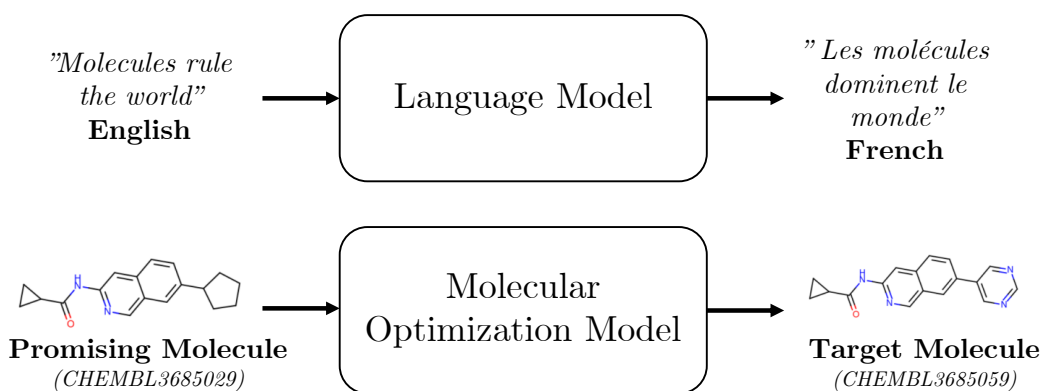


Figure 1.1: Comparison of machine translation and molecular optimization. When a promising molecule is represented as a string, it can be *translated* to a similar molecule with optimized properties using the Molecular Optimization model.

1.3 String Based Molecular Representation

When working with molecules, a common way to describe them is through their *molecular formula*. The molecular formula is a string representation of the molecule which contains information about what atoms the molecule consists of. As an example we have H_2O , which is the molecular formula for water, and it states that water consists of two hydrogen atoms and one oxygen atom. This is an easy representation of a molecule which tells us a lot, but one thing that it does not tell us is how these atoms are connected to each other, in other words, it does not give us any structural information.

To include the structural information, while keeping it as a string, which is suitable if we want a program to read it, one can use the representation *Simplified Molecular Input Line Entry System*, or in short, SMILES [12]. With SMILES, any molecule

can be represented as a string of ASCII-symbols. It is similar to the molecular formula, but extra symbols are added between the atoms to show how they are connected. One example of a SMILES-string is shown in Figure 1.2, together with its corresponding structural formula. In the standard version of the SMILES specification, any molecule can be represented as a string, but the representation is not unique, meaning that multiple strings can correspond to the same molecule. This can easily be corrected for if needed, making each molecule have one and only one string-representation. That representation is referred to as the *canonical* SMILES for the molecule [12].

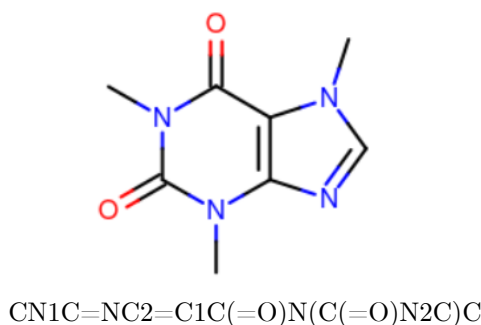


Figure 1.2: The structural formula and SMILES-string for the molecule *caffeine*.

1.4 Project Description: Extensions of Previous Model

In this thesis the aim is to extend and improve the Transformer based model introduced by [1]. This improvement will be in two aspects. The first one is to apply *curriculum learning* [13] to the model. This is a way of training a model through “starting small”, meaning that the model only trains on an easy subset of the data, to later incorporate difficult data. This is with the hopes of it making the training of the model faster and the accuracy better. The second aspect that we will cover, *Core-Fixed formulation*, will be about only optimizing a specific part of the molecule, leaving the other part unchanged under the transformation.

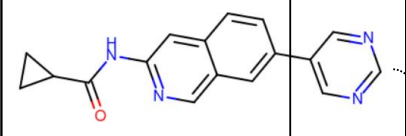
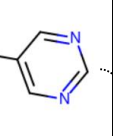
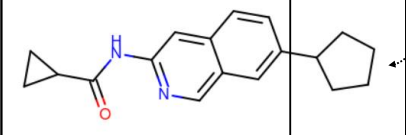
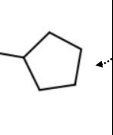
1.4.1 Curriculum Learning

Humans and animals tend to learn better when the content to be learned is presented in a meaningful order, a *curriculum*, usually starting with easy and basic concepts, to later move on to more advanced ones. Motivated by this, research has been performed to see if something similar also holds for the learning of machines. The idea was popularized by Bengio in 2009 [13], but was investigated already in 1993 [14]. After this, studies have been done where curriculum learning has been successfully applied in computer vision [15], and more recent studies have also successfully applied it to language models [16, 17, 18, 19].

In these studies, many advantages of utilizing the concept of a curriculum are mentioned, with the most frequent being that it can increase learning speed and improve final performance on test data by finding a better local optimum from the same training data. Other advantages mentioned are that it can help in generalization and that it gives universal performance improvements on a wide range of natural language understanding tasks.

1.4.2 Core-Fixed Molecular Optimization

The original model produced in [1] was trained to *translate* the entire SMILES of the source molecule into the entire SMILES of the generated target molecule. A disadvantage of this approach is that there is no guarantee that the generated target molecule will be a matched molecular pair with the source molecule, i.e. that the source and generated molecules differ only by a single transformation. With the chemist’s approach, one could be interested in keeping one part of the molecule (core), while exchanging the other (R-group) (Figure 1.3), in order to accomplish a molecule with optimized properties. Motivated by this we will train a model, identical to the original model, except that it will generate only the target R-group instead of the whole molecule, given a source core and R-group. Since the SMILES sequence of the target R-group is shorter than the SMILES sequence of the entire target molecule, such a model might yield shorter training time and improved performance over the original model.

Source Molecule (CHEMBL3685029)	LogD: 4.14 Solubility: -0.01 Clearance: 1.68		
Target Molecule (CHEMBL3685059)	LogD: 2.75 Solubility: 0.72 Clearance: 1.21		
	Properties	Core	R-Group

Transformation

Figure 1.3: An example of matched molecular pair and corresponding properties.

1.5 Thesis Outline

In this introductory chapter, we have given a brief context of this thesis. In Chapter 2, we introduce the Transformer model and give a summary of the previous work which this thesis is built on. We also give a conceptual background to curriculum learning, which we use in one of the extensions we are considering. In Chapter 3, we present the method for accomplishing our two considered extensions curriculum learning and Core-Fixed formulation. This includes preparation of the data and how each model is trained and evaluated. In Chapter 4 we review the results, which the

discussion in Chapter 5 is built on. Finally, Chapter 6 gives the main conclusions of the work.

2

Theory

In this chapter we describe the Transformer architecture, and give a summary to the specific Transformer model that our two extensions are based on. We also give a conceptual background to curriculum learning.

2.1 Transformer

The Transformer architecture was introduced by [2] and was shown to yield state-of-the-art performance within NLP. Through comparison of the BLEU-score, a metric describing the quality of translating a sequence between languages [20], the article showed that the Transformer outperformed its alternatives when translating between English and German. Besides the typical NLP tasks, the Transformer has been shown to perform in other sequence tasks, such as time series forecasting [21].

The input and output of the Transformer consist of token sequences. In the area of NLP the input could represent a sentence in one language that is fed into the Transformer which then generates the sentence's translation in a different language. The Transformer architecture consists of an *encoder* and a *decoder* stack, visualized in Figure 2.1. The encoder stack could be thought of as a mapping from a sequence of tokens in the token space, to a sequence of continuous representations. With the sequence of continuous representations from the encoder, the decoder generates one token at a time based on the previously generated tokens. Note that both the encoder and decoder stacks take each token's sequence position into account using a so called *positional encoding*.

2.1.1 Token Embedding

To numerically deal with the typically text based tokens, one early popular approach is *one-hot-encoding*. One-hot-encoding consists of letting each possible token be represented by a N_{Token} -dimensional vector of $N_{Token} - 1$ '0's and a single '1', where N_{Token} represents the size of the token space. By construction the position for which the '1' occurs will be unique for each token, see (2.1).

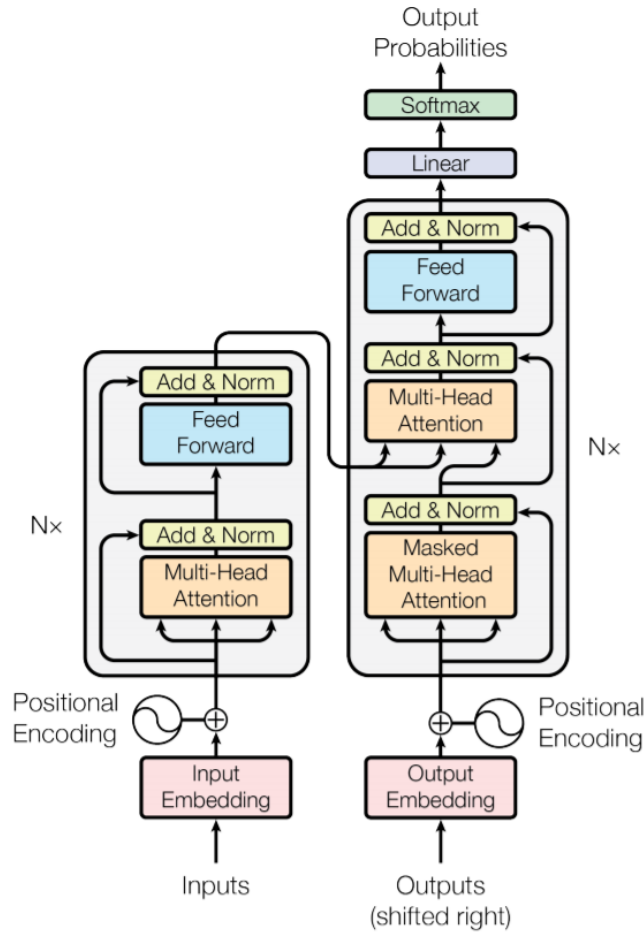


Figure 2.1: The Transformer architecture introduced in [2]. The left part represents the encoder stack and the right part the decoder stack.

$$\begin{aligned}
 \text{Token 1} &: [1 \quad 0 \quad \cdots \quad 0] \\
 \text{Token 2} &: [0 \quad 1 \quad \cdots \quad 0] \\
 &\vdots \\
 \text{Token } N &: [0 \quad 0 \quad \cdots \quad 1]
 \end{aligned} \tag{2.1}$$

The computational complexity is reduced by employing an Input Embedding which limits the dimensionality of the input data presented to the Transformer. Here the Input Embedding is trained together with the rest of the model. Using this embedding, it is further possible for the Transformer to exploit structural similarity between tokens, as can be seen in language applications where representations of similar words commonly lie close to each other in the embedded space. The Input Embedding converts each token from the token space into a continuous d_{model} -dimensional vector.

2.1.2 Positional Encoding

For the Transformer to understand the order of the tokens in the inputs, *positional encoding* is necessary [2]. This is handled by, for token at position pos , adding

$$\text{PE}(pos, i) = \begin{cases} \sin\left(\frac{pos}{10000^{i/d_{model}}}\right), & \text{for } i \text{ even} \\ \cos\left(\frac{pos}{10000^{(i-1)/d_{model}}}\right), & \text{for } i \text{ odd} \end{cases} \quad (2.2)$$

elementwise for $i \in [1, d_{model}]$ to the continuous representation vector yielded by the input embedding.

2.1.3 Attention

As seen in Figure 2.1, both the encoder and decoder stacks of the Transformer relies on *Multi-Head Attention* layers. A Multi-Head Attention layer consist of multiple so-called *Scaled Dot-Product Attention* heads.

Scaled Dot-Product Attention A Scaled Dot-Product Attention head uses three matrices W^Q , W^K and W^V of dimensions $d_{model} \times d_k$, which are learned during model training. Here the hyperparameter d_{model} comes from the dimension of the Input Embedding, and d_k is a hyperparameter for the attention layer. With W^Q , W^K and W^V , the query matrix Q , key matrix K and value matrix V are calculated using

$$\begin{aligned} Q &= XW^Q \\ K &= XW^K \\ V &= XW^V, \end{aligned}$$

where row i in X equals the (current) continuous representation of token i . Letting n correspond to the number of tokens, X would thus be of dimension $n \times d_{model}$. Using Q , K and V the Scaled Dot-Product Attention is calculated,

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \quad (2.3)$$

where softmax is the activation function defined, for a vector index j , through

$$\text{softmax}_j(\mathbf{z}) = \frac{e^{z_j}}{\sum_{k=1}^{d_{model}} e^{z_k}}.$$

The usage of the scaling factor $\frac{1}{\sqrt{d_k}}$ in (2.3) is intended to counteract the risk of obtaining small gradients during back-propagation. [2]

Multi-Head Attention With the usage of multiple Scaled Dot-Product Attention heads we obtain Multi-Head Attention. Letting \mathbf{Z}_{HEAD_i} represent the output from Scaled Dot-Product Attention head i , the Multi-Head Attention is obtained by

$$\text{Multi-Head Attention} = [\mathbf{Z}_{HEAD_1} \quad \mathbf{Z}_{HEAD_2} \quad \cdots \quad \mathbf{Z}_{HEAD_h}]W^O,$$

where h corresponds to the number of Scaled Dot-Product Attention heads and W^O is a $hd_k \times d_{model}$ matrix that is learned during training. Note that each head i is associated with unique W_i^Q , W_i^K and W_i^V matrices, and that the hyperparameter d_k , describing the dimensionality of these, is shared among the heads. With W^O , the Transformer model can learn how to weigh the outputs from the different Scaled Dot-Product Attention heads, where each head can be thought to capture different qualities of the token sequence.

2.1.4 Encoder

The encoder stack consists of multiple stacked encoder layers, where the input to the first layer is the continuous representation after the positional encoding mechanism. Each encoder layer itself consists of two sub-layers, firstly Multi-Head Attention - described in the previous section - and secondly a positionwise fully connected feed-forward network. The feed-forward network in a specific encoder layer is represented by

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2, \quad (2.4)$$

which we recognize as two linear transformations with a ReLU activation in between. Note that the **max** function works element wise on the vector $xW_1 + b_1$. To match the input and output dimensions, the W_1 matrix is of dimension $d_{model} \times d_{ff}$, the vector b_1 of dimension d_{ff} while matrix W_2 is of dimension $d_{ff} \times d_{model}$ and vector b_2 is of dimension d_{model} . Here d_{ff} represents an additional tuning parameter.

Both sub-layers are used within a residual connection, which in token i yields the output y_i from the input x_i according to

$$(y_i)_j = \frac{(x_i + \text{Sublayer}(x_i))_j - \mu_j}{\sigma_j},$$

where j is the component and Sublayer represents the function of a sub-layer (i.e. FFN or Mult-Head Attention), and where

$$\begin{aligned} \mu_j &= \frac{1}{n_{token}} \sum_{i=1}^{n_{token}} (x_i + \text{Sublayer}(x_i))_j, \\ \sigma_j &= \sqrt{\frac{1}{n_{token}} \sum_{i=1}^{n_{token}} ((x_i + \text{Sublayer}(x_i))_j - \mu_j)^2}. \end{aligned}$$

Here n_{token} represents the number of tokens in our input sequence. This technique is known as *layer normalization* and was introduced for sequence based neural networks in [22] to reduce training time. This computation is represented by *Add & Norm* in Figure 2.1.

2.1.5 Decoder

Like the Encoder, the Decoder stack of the Transformer network consists of a stack of N decoder layers. Each decoder layer employs similar sub-layers as the ones of the

encoder. Apart from the Multi-Head Attention sub-layer and the feed-forward network, the decoder also uses an additional Multi-Head Attention sub-layer, denoted by *Masked Multi-Head Attention* in Figure 2.1. The Masked Multi-Head Attention works on the token sequence that have been generated so far. For this Multi-Head Attention to not use output tokens that have not been generated yet the model fixes the weights of W^Q , W^K and W^K to $-\infty$ corresponding to the (as of yet) not generated tokens. Note that every sequence that is generated by the decoder, starts with a common *start token*, which is treated by the decoder as if it was the first generated token.

Following the Masked Multi-Head Attention sub-layer is another Multi-Head Attention sub-layer which takes both the output from the previous decoder layer, as well as the output of the encoder, as input. Like with the encoder, the final layer in the decoder is a feed-forward network as defined in (2.4). For each decoder sub-layer the Transformer also employs the layer normalization, much like the encoder.

2.1.6 Sequence Generation

The output of the decoder stack is fed into a single linear layer which yields a vector of dimension d_{model} . That output vector is then fed into a softmax function which converts the continuous vector to output probabilities in the output token space. The output probabilities are used to generate a tokens, typically with one of the following techniques:

Greedy Decode - The token with highest probability is chosen. After a token has been chosen the decoding procedure repeats to select the next token with the highest probability given the previous ones. Note that this choice of decoding procedure is *deterministic*, meaning that the model will always generate the same sequence for a given input.

Multinomial Decode - A token is sampled based on interpreting the token probabilities as a discrete distribution. As for greedy decode, the decoding procedure is repeated to select the next token given the previous ones, after a token has been chosen.

Beam Search - Using the beam width B , token probabilities are generated for each of the B most likely token sequences from the previous step. What determines the (model-estimated) conditional probability of a token sequence is the product of each of the tokens given the previous ones, i.e.

$$P(t_0, \dots, t_{N-1} | \mathbf{x}) = P(t_0 | \mathbf{x}) \prod_{i=1}^{N-1} P(t_i | \mathbf{x}, t_0, \dots, t_{i-1}),$$

where \mathbf{x} constitutes the input sequence, t_j for $j \in [0, N - 1]$ represents the token at position j and N the length of the token sequence. When B sequences have been generated, the token sequence with the highest conditional probability is chosen.

Note that for the model to start and stop generating tokens appropriately, common *start* and *end* tokens are used. The *start* token is used as the first generated token in the Outputs in the Output embedding, see Figure 2.1. The *end* token is used by the model to determine when to stop generating new tokens.

2.2 Transformer for Molecular Optimization

The article [1] showed potential for the Transformer within molecular optimization, where a *source* molecule is transformed in to a similar molecule, called *target* molecule with new, desirable *molecular properties*, defined as input together with the source molecule. What molecular properties where considered is explained in the following section.

2.2.1 Considered Molecular Properties

As a proof of concept, the Transformer developed at AZ is considering the three ADMET properties *LogD*, *Solubility* and *Clearance*:

LogD - LogD which is short for the logarithm of the *partition coefficient*, is a measure of the lipophilicity of the molecule, or in other words its potential to dissolve in lipids, fats and other non-polar solvents. This is relevant in drug discovery since a molecule with high lipophilicity is more likely to penetrate cell membranes [23]. A too high LogD could, however, be toxic.

Solubility - Generally, Solubility is a measure of a molecules capacity to dissolve in a certain solvent. In this project, as in [1], we take the solvent to be water. Molecules that have high Solubility (in water) tends to be lipophobic, meaning that they have low potential to dissolve in lipids, fats and other non-polar solvents. This means that this measure gives us similar information as the *LogD* measure, but in a hydrophilic manner instead.

Clearance - Clearance is a measure of how fast the substance will be removed from the patient’s body, giving essential information related to metabolic stability and dosing of the drug.

2.3 Model Description

The original Transformer model for molecular optimization was trained on a subset of the MMPs extracted from ChEMBL. Each MMP was represented by source and target molecules as their SMILES-strings. The input of the model consisted of source SMILES-string concatenated with *property-change-tokens*, which encodes the desirable property changes in the previously mentioned molecular properties *LogD*, *Solubility* and *Clearance*, while the output consisted of the entire SMILES-string of the target molecule. To represent these property changes as a finite set of tokens, each relevant property change was assigned a category represented by a unique token.

Figure 2.2 shows an example of an input and output of the Transformer model in [1].

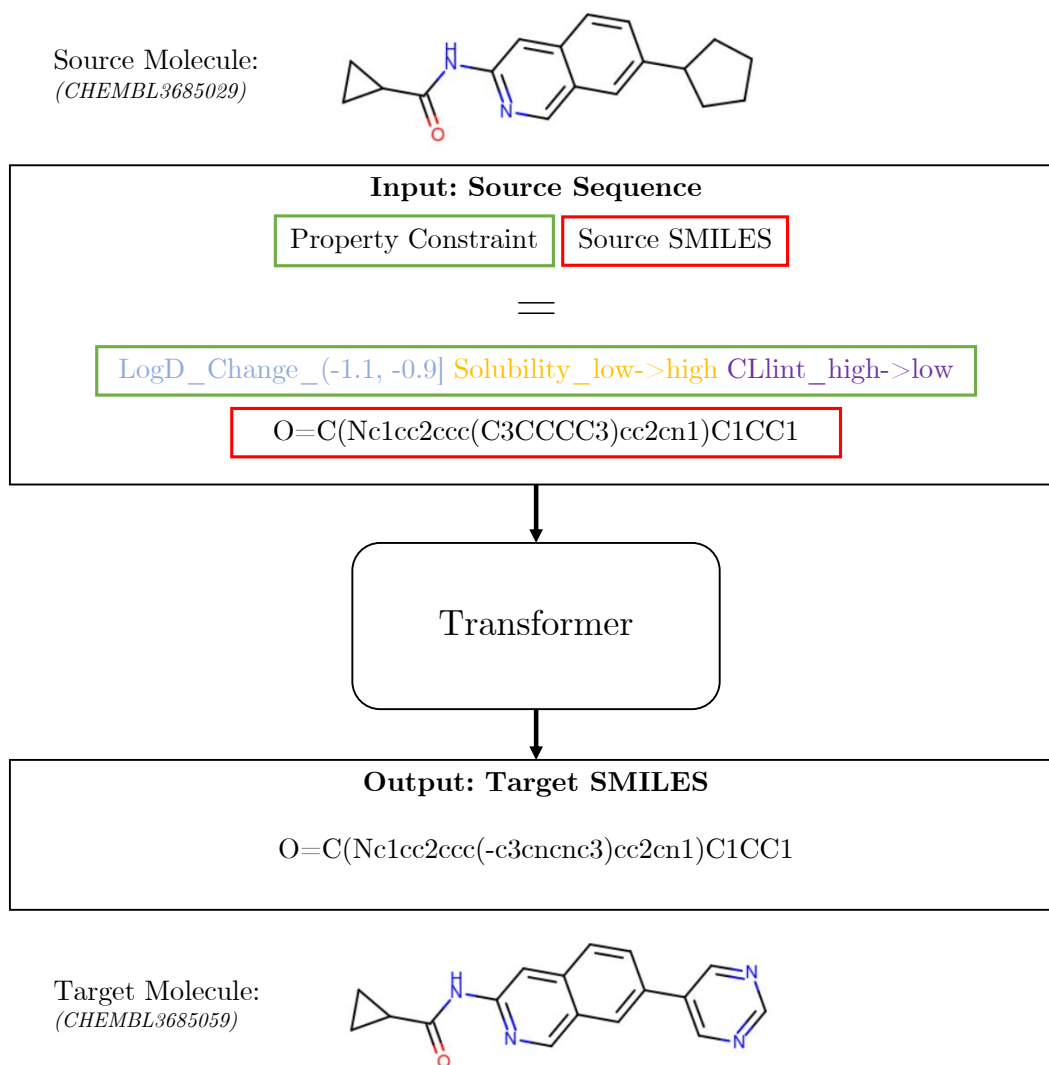


Figure 2.2: An example of input and output of the previous model [1].

With the training set D , the model was trained to minimize the Negative-Log-Likelihood

$$\text{NLL}(\theta) = - \sum_{i \in D} \sum_{t=1}^{|\mathbf{y}_i|} \log P(\hat{y}_{i,t} = y_{i,t} | y_{i,1}, \dots, y_{i,t-1}, \mathbf{x}_i; \theta), \quad (2.5)$$

where θ represents the model weights and \mathbf{x}_i the input sequence i . Here $|\mathbf{y}_i|$ represents the target length and $y_{i,t}$ the token at position t for the target i . Note that $\hat{y}_{i,t}$ represents the generated token at position t , conditioned on the model weights, input and true previous tokens $y_{i,t}$. This technique of conditioning the generation of the next token based on the true previous tokens, rather than the previously generated ones, is known as *teacher-forcing* [24].

The updating of the Transformer weights through backpropagation used the ADAM-optimizer [25] with learning rate 0.0001 along with a batch size of 128. Furthermore, the internal neural networks used a dropout [26] of 10%. After each training epoch, i.e. one pass through the entire training set, the model was validated using (2.5) with the previously mentioned validation set to confirm generalization improvement over the last epoch. The model was trained for a total of 60 epochs after which it was compared to two alternative sequence generation models; based on Seq2Seq [27] and HierG2G [28]. The evaluation was based on letting the models generate 10 target molecules for each input in the test set, consisting of SMILES sources concatenated with property change tokens. For the Transformer to accomplish a range of generated molecules the model used multinomial decode, described in Section 2.1.6, as it was shown to yield similar performance to the more computationally heavy but typically more accurate Beam-search.

To estimate the properties of the generated molecules, and thus validate which molecules satisfy the specified property changes, AstraZeneca’s internal property prediction models were used. The Transformer model outperformed its alternatives in terms of how many of the generated molecules that satisfied the desired property changes.

The hyperparameters that were used for the Transformer in [1] are seen in Table 2.1.

Table 2.1: Hyperparameters for the Transformer used in [1]

Parameter	Value	Description
d_{model}	256	Dimension of continuous representation for the input and output encoding
d_{ff}	2048	Dimension for W_1 and W_2 in the fully connected neural network in (2.4)
d_k	64	Number of rows in Q , K and V for the attention mechanisms
N	8	Number of layers for the Encoder and Decoder
h	6	Number of heads for the attention mechanisms in the encoder and decoder

2.4 Curriculum Learning

The idea with curriculum learning is to “start small”, meaning only to train on a subset of the input-output pairs in the training dataset that is *easy* in earlier epochs, and to incorporate more *difficult* data in later epochs. This is opposed to the usual method of training on all available training data in each epoch.

2.4.1 Difficulty Assessment

To define a curriculum, a difficulty score is used. The difficulty of an input-output pair does not have a unique definition and can be assessed in multiple ways, but is supposed to reflect how difficult it is for the model to learn from it. One example of an assessment of the difficulty when training a model for language translation from one language to another could be to assess the difficulty of an input-output pair as the number of words in the input sentence [18], meaning that an input-output pair with input sentence “Hi” would have a difficulty of 1, and if it instead would be “I am home” it would have a difficulty of 3. The actual number stating the difficulty here is a bit arbitrary, and it is only the induced order of the input-output pairs that we need. Other ways to put a difficulty score on each pair can be by considering the frequency of the pair or its parts in the training data. In the case of sentences, this could be that the more uncommon words there is in a sentence, the higher the difficulty score would be for it [18].

The difficulty assessments discussed above is what could be called *human heuristic based*, meaning the difficult is based on what we as humans consider difficult. Another way of doing this, which could be called *machine based*, is to decide the difficulty automatically by looking at what data the model learns easily from and not, e.g. the loss induced by the training data. In this project we only investigate the *human heuristic based* difficulty assessments.

2.4.2 Curriculum Arrangement

When the difficulty score is set for each input-output pair, the pairs are divided into N **Buckets** $C_i, i \in 1, 2 \dots N$. The first bucket C_1 only includes a subset of the pairs that have the lowest difficulty score. Consecutive buckets after this include data that is more difficult than what was in the previous bucket, and this procedure goes on until all data have been distributed across the buckets. The number of buckets N in the curriculum, as well as the number of data points in each bucket, are sometimes naturally taken from the difficulty scores. In the example of when the difficulty was assessed by the length of the input sentence, it is natural to have one bucket for each available length in the training data, where each bucket would contain all data that has the corresponding length. This can in general be chosen more arbitrarily as well.

With these buckets, the training will be split up into steps, one step per bucket, where each step progress for some number of epochs. In step i , the data used for training will be all the buckets up to the i :th bucket, or more formally, the data used in training step i is $C_1 \cup C_2 \cup \dots \cup C_i$. Figure 2.3 visualizes this split into buckets and what data is used in each step of the training. Worth noting here is that since each consecutive step includes more data, one epoch in an early step will most likely take less time to go through, but also since those steps use less data, the model will most likely learn less in one epoch and thus need more epochs to converge.

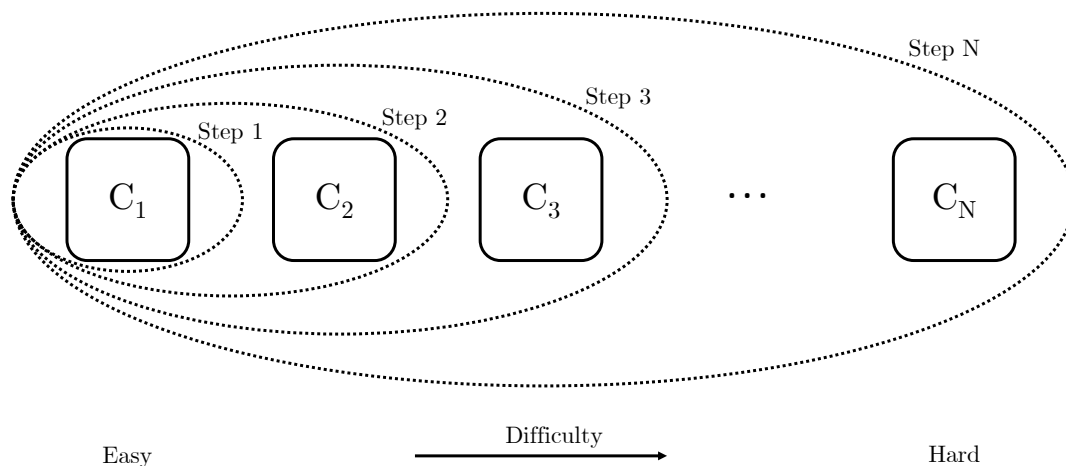


Figure 2.3: Visualization of difficulty based buckets in curriculum learning. Easy data is put into some buckets and difficult data into others. The training is done by starting training with only the easy buckets, later to incorporate more difficult ones.

The potential advantages of utilizing curriculum learning is discussed in Section 1.4.1. Based on that discussion, we decide on looking at the two main potential advantages mentioned there, namely:

- If it can result in shorter training times
- If it can improve final performance on test data

These advantages will be investigated within molecular optimization using various manually defined curricula, as we will come back to later.

3

Methods

In this chapter we will focus on introducing the experimental settings used to evaluate the models in further chapters. We will go into detail on how we train and evaluate the models, including how we create and work with data.

3.1 Platform

The implementations for preprocessing, training, generating and evaluating the described experiments are written in Python. Our implementations are highly based on the ones used for accomplishing the experimental results in [1]. The Transformer model is built using PyTorch, based on Torch, with inspiration from [29]. The GPU used for training the Transformer models was a Tesla V100.

3.2 Data Preparation

In this section we present the data that will be used, including how it is preprocessed and how the molecular properties are represented for the models.

3.2.1 Preparation of Matched Molecular Pairs

To begin with, we will work with matched molecular pairs (MMPs) extracted from ChEMBL [30]. This was done using an open-source matched molecular pair tool [31]. The molecules were then standardized using the molecule validation and standardization tool MolVS [32]. Note that this constitutes the same molecule pairs as were used in the original Transformer for molecular optimization [1]. To make the model work with high-quality and drug-like compounds, the original Transformer used a subset of the molecular pairs satisfying a number of constraints, which are stated below:

- The number of heavy atoms, i.e. any atom except hydrogen, of the core is less than 50
- The number of heavy atoms in R-group is less than 13
- The ratio of heavy atoms in the R-group to the entire molecule is less than 0.33
- The number of H-bond donors in the R-group is less than 3
- The number of H-bond acceptors in R-group is less than 3
- AstraZeneca’s AZFilter “CORE” [33] to filter out bad-quality compounds

- Each molecule’s property values are within 3 standard deviations of all molecules’ property values

ChEMBL provides 9,927,876 molecular pairs for which the constraints are satisfied. Out of these we will use a 2% random selection to limit the training times. Among these we will use 81% for training, 9% for validation and 10% for testing. In absolute numbers, this corresponds to 160,831 pairs, 17,871 pairs, and 19,856 pairs respectively.

3.2.2 Property Representation

For each molecule in a molecular pair, the three ADMET properties *LogD*, *Solubility* and *Clearance* are calculated using property prediction models developed in [1]. These models are based on message passing neural networks [34] trained on in-house experimental data. For each MMP, the changes in the molecule’s property values are calculated, which are lastly encoded as discrete property change tokens. Considering practical desirable criteria and experimental errors, the change in *Solubility* and *Clearance* are encoded into one out of three tokens, while the change in *LogD* is encoded into one out of 60 tokens representing different value intervals.

Each MMP is assigned the tokens which correspond to the intervals associated with the property changes. For *LogD* the token will represent the interval for which the *LogD*-change lies within. For *Solubility* and *Clearance* respectively, there are only three intervals/tokens. If the value (*Solubility* or *Clearance*) is higher in the second molecule of the pair, the pair gets the token “low->high” for that property, if the value is higher in the first molecule of the pair, the pair gets the token “high->low”. Since there will be an uncertainty in the individual molecules’ property change estimates, *Solubility* and *Clearance* have fixed thresholds which are used to determine if the change is significant. For absolute changes lower than the corresponding property threshold the pair is assigned the token “no change” for that property. Figure 3.1 shows all the possible property tokens.

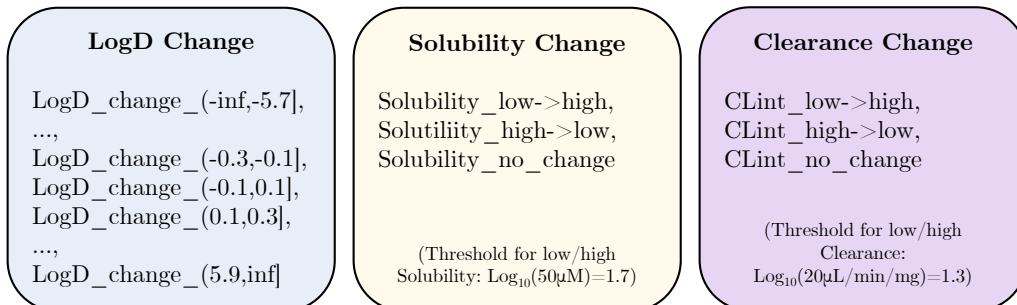


Figure 3.1: All encoded property change tokens. Thresholds for high/low Solubility and Clearance are given in the figure.

3.3 Curriculum Learning

In the following sections the approach to accomplishing models based on training using curriculum learning is presented. This includes the various types of curriculum learning settings that are considered, and how the resulting models are trained and evaluated.

3.3.1 Simulation of Missing Property Values

As stated earlier, each molecule has a value for the three different properties *LogD*, *Solubility* and *Clearance*, which are all estimated using internal property prediction models. The accuracy for the estimates is not optimal however, and in practice the properties will be measured experimentally. When using the available experimental data, it is not always the case that all the properties have been measured, e.g. some molecules only have *LogD* measured but not *Solubility* or *Clearance*. In the previous way of training the model, a molecule that is not complete, i.e. misses at least one of the property values, will not be able to be used at all in training, although most of the information is still there. With this in mind, we start off by *masking* our data, meaning that we remove some of the property values for the molecules. This makes it more similar to the practical scenario. Furthermore, we adapt the implementation to only include the property change tokens for which the actual property change is available for a specific MMP.

Note that this procedure of masking would also make sense to do in the other part of the project, Core-Fixed molecular optimization, but we decided to use the original data there, to make it easier to compare to the original model. Another advantage of doing this in the context of curriculum learning is that we get another natural way of assessing the difficulty, namely by defining the difficulty by the number of available properties, which we will come back to later.

The molecules were masked according to the following procedure: Firstly all unique molecules, out of both source and target molecules were listed, then **20%** of the molecules got their *LogD*-value masked, **30%** of the molecules got their *Solubility*-value masked and **40%** of the molecules got their *Clearance*-value masked. The choice of what molecules were masked was random, and independent on the masking of the other properties. Figure 3.2 shows all different combination of properties that a molecule can have after the masking as an Venn-diagram, with areas proportional to the amount of molecules in each set. A molecular pair will then have a certain property masked if at least one of the molecules it constitutes of has it masked. Table 3.1 shows the number of each type of molecule after masking, where each number in the “molecule” column corresponds to a closed area in the Venn-diagram.

Since 0.8 of the molecules have a *LogD*-value and 0.7 of the molecules have a *Solubility*-value, and since the masking of a property was independent of the masking of another property, the amount of molecules that have both *LogD* and *Solubility* will be approximately $0.8 \cdot 0.7 = 0.56$, which is what we see in Table 3.1. And for a

molecule *pair* to have a *LogD*-value, both of the molecules that it consists of needs to have it, which they do with a probability of $0.8 \cdot 0.8 = 0.64$. This correspondence is also seen in Table 3.1. With similar reasoning, all the (expected) proportions shown as percentages in Table 3.1 can be calculated.

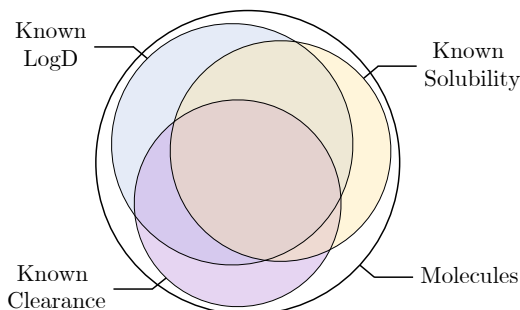


Figure 3.2: Visualization showing how the data was masked to simulate practical scenario of missing properties.

Table 3.1: The amount of molecules and molecular pairs still having its properties after the masking. Each row corresponds to one region in the Venn-diagram shown in Figure 3.2.

	Molecules	Molecular Pairs
Total	248,346 (100%)	198,558 (100%)
LogD	198,677 (80%)	127,336 (64%)
Solubility	173,843 (70%)	97,586 (49%)
Clearance	149,008 (60%)	71,322 (36%)
LogD and Solubility	139,143 (56%)	62,637 (31%)
LogD and Clearance	119,162 (48%)	45,614 (23%)
Solubility and Clearance	104,213 (42%)	34,988 (18%)
LogD, Solubility and Clearance	83,349 (34%)	22,324 (11%)

3.3.2 Difficulty Assessment

An important part of designing a curriculum is the difficulty assessment. In this project we will explore three different kinds: **Property-Based**, **Length-Based** and **Token Rarity-Based**. We will also compare these to a trivial, random assessment which we will call **Random-Based**, and an assessment where we flip the order of a previous assessment, called **Reverse**.

Exploratory Difficulty Assessments:

By **Exploratory Difficulty Assessments**, we mean the following 3 difficulty assessments.

- **Property-Based** - This difficulty assessment will use the fact that we now have masked some of our property values (see above) and assigns the difficulty

for an input-output pair as the number of available properties, which is an integer between 0 and 3. This difficulty assessment is based on the assumption that a bigger number of available property change tokens are harder to learn.

- **Length-Based** - This difficulty assessment assigns the difficulty for an input-output pair as the number of tokens that the input molecule’s SMILES-representation constitutes of, which is an integer between 10 and 77 in our dataset. This difficulty assessment is based on the hypothesis that longer sequences are harder to learn.
- **Token Rarity-Based** - This difficulty assessment assigns the difficulty for an input-output pair first by listing all SMILES related tokens, sorted by frequency in the training set, meaning that the frequent tokens will be first and rare tokens will be last. An input-output pair will then get its difficulty score as the position in the sorted list of its most rare token, which is an integer between 6 and 30 in our dataset. This difficulty assessment is based on the hypothesis that inputs with more rare tokens are harder to learn.

Comparative Difficulty Assessments:

By **Comparative Difficulty Assessments**, we mean either the **Random-Based** difficulty assessment or any assessment defined as a **Reverse**. In this project, we only apply **Reverse** on the previously mentioned **Property-based** difficulty assessment.

- **Random-Based** - This difficulty assessment assigns the difficulty for an input-output pair randomly. This difficulty assessment will only be used for comparison purposes.
- **Reverse** - For each difficulty assessment, one can also define its **Reverse** assessment, which flips the list ordered by difficulty upside-down, taking the easy input-output pairs as difficult, and vice versa. This will also only be used for comparison purposes. If the Exploratory difficulty assessments are indeed good ones, then applying Reverse on one of them should result in a bad difficulty assessment.

3.3.3 Curriculum Arrangement

After the difficulty assessment, the choice of how many curriculum learning steps and how much data should be in each step needs to be made. For Property-Based assessment, the division that is used follows naturally from the difficulty assessment: 4 steps are used, and the data in step i will be the data that has i properties available.

For Length-Based and Token Rarity-Based assessment, a natural division would also be to have one step for each difficulty level, which would be 66 and 24 steps respectively, as were seen in the previous section. But to make our different methods

3. Methods

more comparable, we use 4 steps here as well. For Length-Based, Token Rarity-Based and Random-Based assessment, the data is spread uniformly in the buckets, meaning that all buckets have approximately equally many input-output pairs. This is done by firstly splitting the pairs into the quartiles (which there are exactly 4 of) based on the score given by the chosen difficulty assessment, and secondly assigning the first bucket the first quartile, the second bucket the second quartile and so on. The reason for it being approximately uniform is that the split is not made exactly at the quartiles, but at the closest place in the list where the difficulty score changes. For Length- and Token Rarity-Based difficulty, using 8 steps will also be considered with the same percentile-based data split as for 4 steps. Figure 3.3 shows how the data is split up based on difficulty scores for three of the difficulty assessments.

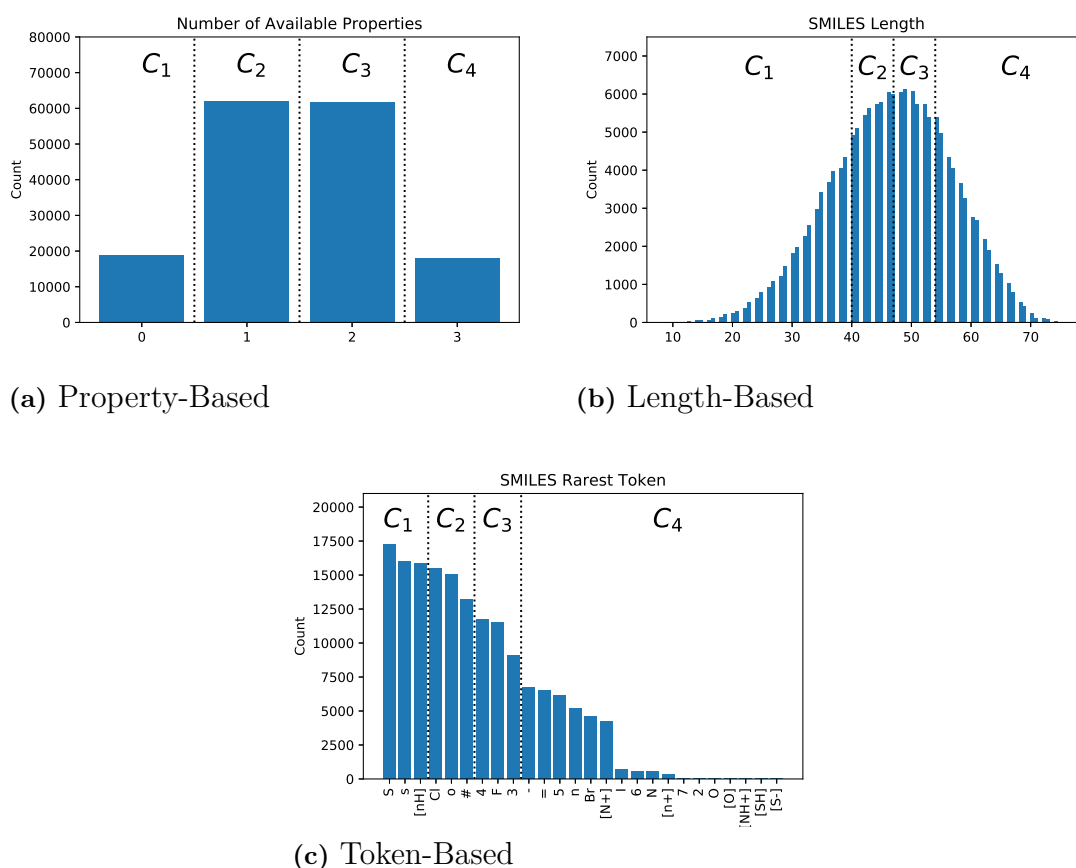


Figure 3.3: Training data division for Property-, Length- and Token Rarity based difficulties with 4 buckets. In each graph C_i represents bucket i .

3.3.4 Baseline

For curriculum learning, one baseline will be considered. The baseline will be the Transformer model described in [1], except that it is trained on the masked data described in Section 3.3.1. This will be compared to the curriculum learning approach, where more data is introduced in steps. The idea with the baseline is to investigate whether curriculum learning benefits the training.

3.3.5 Training and Validation

Through curriculum learning the training is structured in steps according to the various difficulties (curricula) presented in Section 3.3.2. For each curriculum learning step the model will be trained for a predefined number of epochs, which could be considered as hyperparameters. As for the original Transformer for molecular optimization we will consider the training through minimization of the Negative-Log-Likelihood, described in Section 2.2. Note, however, that unlike the original model, the used training set will depend on the current curriculum learning step.

To measure the models' improvement over training, a common set of molecular pairs designated for validation (Section 3.2.1) is used. Among all the 17,871 molecular pairs aimed for validation, only the ones with (simulated) three properties are considered, resulting to a size of 1,984. This is due to the general aim of generating molecules based on the three property changes, as was noted in Section 3.3.2.

For the validation set, the Negative-Log-Likelihood loss and the validation accuracy are calculated after each epoch. The validation accuracy represents the proportion of molecules generated from the validation set using greedy decode, described in Section 2.1.6, that are equal to the (true) validation target molecules. Since the validation- and training sets are expected to possess similar data qualities by construction, and as the true target molecules fulfill the desirable molecular properties, these validation measures are reasonable.

For the model- and training hyperparameters we will use the same ones as presented in Section 2.2 for the original Transformer for molecular optimization. This is done since it is reasoned that both models work in a similar input- and output space and could be expected to take similar benefits from the specific model architecture.

3.3.6 Test Sets

To evaluate the models' performances in different relevant scenarios, we will use a few different test sets. Since we are interested in the models' performances on molecules with three available properties, we will only use the part of the test data for which the pairs have been simulated to have three properties. This means that the constructed test sets will only use a subset of the 19,856 pairs designated for testing purposes described in Section 3.2.1. During evaluation we will use the test sets below.

Test-Original* - This test set constitutes of all molecular pairs with three properties used for testing purpose, corresponding to a size of 2,229 pairs. For this test set we will use the source molecules, as well as the property change tokens corresponding to the target molecules in the test set.

Test-Molecule* - This test set contains all molecular pairs present in Test-Original* except for the pairs for which the source molecule is seen in the train set. This construction yields a test size of 1,393 molecular pairs. The purpose of this test set is

to verify how well the models generalize on unseen starting molecules.

Test-Property* - This test set constitutes of 912 starting molecules with low Solubility, high Clearance, and LogD between 2 and 4.4 in Test-Original*. For this test set we are interested in achieving lower LogD, high Solubility and low Clearance. The desirable LogD for the target molecules are constrained to lie in the interval 1.0 to 3.4, as the experimental data for which the internal prediction model was trained on, lies in this range. The purpose of Test-Property* is to evaluate the models on how well they perform on the particular property changes we are interested in. Note that this specific property change only corresponds to about 0.2% of the molecular pairs in the training data.

3.3.7 Evaluation Metrics

For comparing the curriculum learning models and their baseline we will look at a few evaluation metrics, highly based on the metrics considered in [1]. The metrics are based on the performance of the generated molecules from each model. The generated molecules come from letting each model generate up to 10 unique and valid SMILES molecules for each source molecule in the test sets. To generate 10 unique and valid SMILES molecules, multinomial decode (Section 2.1.6) is used with up to 100 trials. For the generated molecules, the following measures will be used:

Desirable - This metric gives the proportion of generated molecules that fulfill the desirable properties. Since the goal of the model is to generate molecules with the desirable property values specified as input, this is the most important metric. It gives a number that reflects how good the model is at generating these molecules, and therefore a high *Desirable* value is preferred.

MMP33 - This refers to the proportion of generated molecules for which the ratio between the number of heavy atoms in the R-group and the number of heavy atoms in the entire molecule is less than 0.33. From Section 3.2.1 it is recognized that the training data was constructed in such a way that all matched-molecular-pairs satisfy this. This evaluation metric will give a measure of how well the models have learned to model the data used during training, and therefore a high MMP33 value is preferred.

Novel Transformations - This refers to the proportion of generated molecules which yields a transformation (i.e. specific R-group change) which has not been seen in the training data. This metric will give an idea of the models' generalization performance. Note that the performance interpretation of this metric will depend on the corresponding Desirable and MMP33 values, e.g. many novel transformations are not preferable if a model scores bad in terms of Desirable.

3.4 Core-Fixed Molecular Optimization

In the following sections the approach to accomplishing the Core-Fixed molecular optimization model is introduced. This includes how the input and output to the underlying Transformer model are represented, and how the resulting model is trained and evaluated.

3.4.1 Input and Output Representations

In Core-Fixed molecular optimization we are training a Transformer model to generate the SMILES of the R-group of the target molecule, rather than the whole target SMILES as was seen for the standard Transformer model in [1]. For the Core-Fixed model to distinguish between the core and the R-group of the source sequence, we will modify the input sequence compared to the standard formulation. The input sequence for the Core-Fixed model will consist of the property constraint tokens, the core and the R-group as well as a separator token between the core and the R-group. This can be compared to the standard model [1] which use the property change tokens concatenated with the source SMILES as the input sequence.

The separator token that we will use for the Core-Fixed model will be the dot (“.”) - symbol. The entire input and output chain of the Core-Fixed model is visualized in Figure 3.4, which, for easy comparison, uses the same example molecule as was seen for the corresponding visualization of the standard formulation in Figure 2.2.

3.4.2 Baselines

For the Core-Fixed model we will consider two baselines:

Baseline - For our main baseline we will consider the model developed by [1] that was trained to generate the entire target molecules at once, in contrast to the Core-Fixed formulation.

Enumeration Baseline - This constitutes of an exhaustive algorithm, that, for a test molecule, consists of first enumerating over the seen R-groups in the training data and attaching each to the molecule’s core, secondly, selecting all found R-groups which yielded a molecule with desirable properties. This is seen in Algorithm 1. Note that the Enumeration Baseline is not strictly applicable for comparison to the original Transformer model for molecular optimization since for the original model, the concept of cores and R-groups had not been introduced.

3.4.3 Training and Validation

For the Core-Fixed model we will consider a similar training procedure as for the original Transformer [1], through minimization of the Negative-Log-Likelihood loss, described in Section 2.2. Note, however, that the meaning of what is minimized is different since the target sequence for the Core-Fixed model will represent the

3. Methods

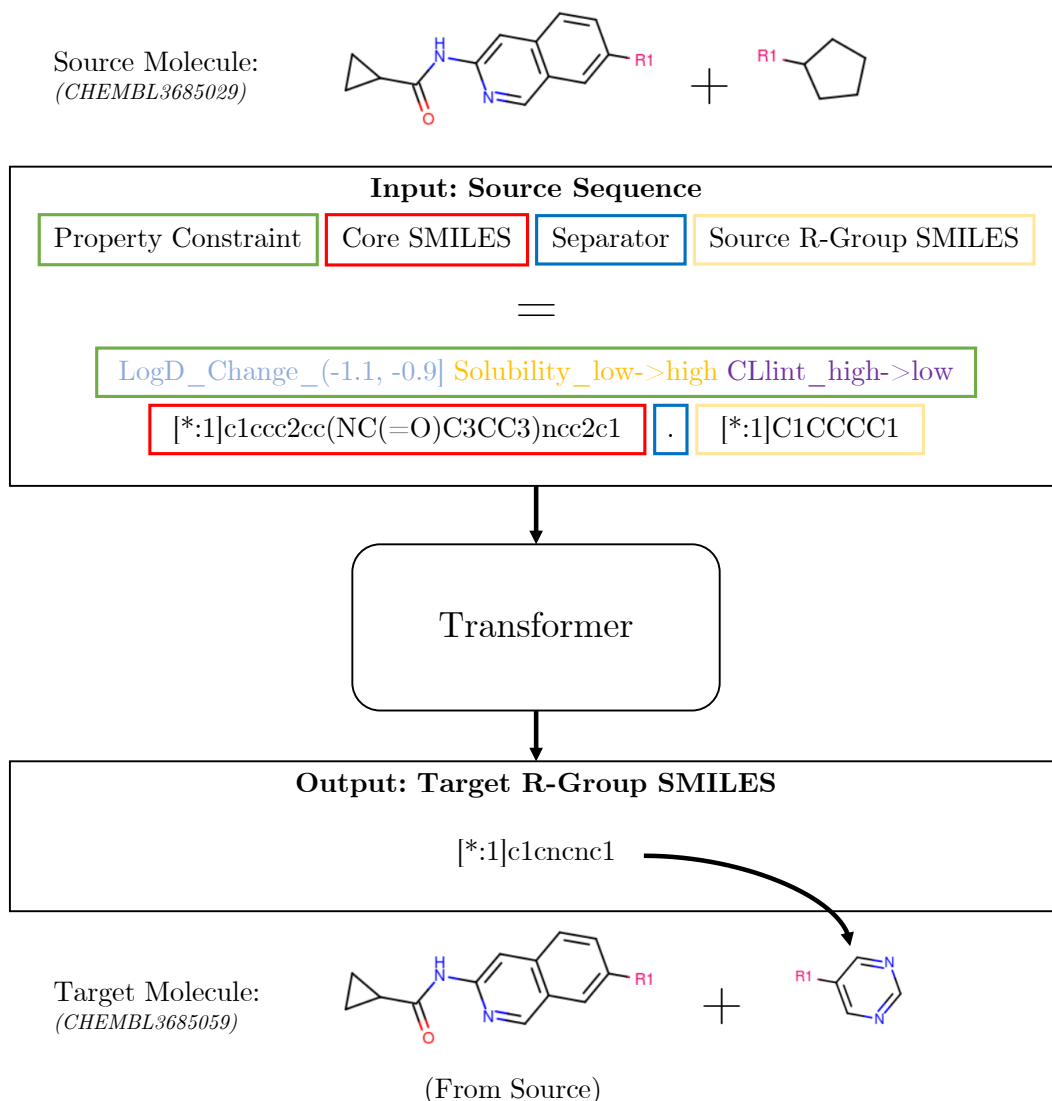


Figure 3.4: Input and output chain of the Core-Fixed model. The input consists of property change tokens, the SMILES of the core, the SMILES of the source R-group and the dot (“.”) -symbol separating the core and R-group representations. The output consists of a single R-group, which, when merged with the core from the source molecule, forms the target molecule.

SMILES of the target R-group, as opposed to the SMILES of the entire target molecule as for the original Transformer for molecular optimization.

As with curriculum learning, the validation set will be used to measure model improvement when training through the Negative-Log-Likelihood loss. For the Core-Fixed model, the validation accuracy represents the proportion of generated R-groups for the validation set using greedy decode, described in Section 2.1.6, that are equal to the (true) validation target R-groups.

During training we will also consider other hyperparameter choices than what was

Algorithm 1 Exhaustive algorithm used to find which of the known R-groups that, when attached to the core c , yields a molecule that fulfills the desired properties p .

```

1: procedure ENUMERATIONBASELINE( $c, p$ )
2:   ( $c$  - core SMILES from the test set)
3:   ( $p$  - corresponding desirable properties)
4:   Let  $R_T = \{\text{All unique R-groups in train set}\}$ 
5:   Let  $V_c = \emptyset$  ▷ Set of R-groups yielding desirable properties
6:   for  $r \in R_T$  do
7:     Let  $M = \text{Concat}(c, r)$  ▷ Concatenate core and R-group
8:     if  $M$  fulfills  $p$  And  $\text{HeavyAtomsRatio}(r, c) < 0.33$  then
9:       Let  $V_c = V_c \cup \{r\}$  ▷ Add  $r$  to  $V_c$ 
return  $V_c$ 

```

used for the original Transformer model described in Section 2.2. Since the output of Core-Fixed model represents a different molecular space, i.e. the space of R-group SMILES as opposed to the space of entire molecules’ SMILES, an optimal model would not necessarily benefit from using the same hyperparameters. R-group SMILES are shorter by construction, but the relationships between the SMILES tokens might also be more complex for a model to learn. With these facts it is hard to get a prior understanding whether the Core-Fixed model would benefit from having a more complex or simpler architecture than the original model [1].

3.4.4 Test Sets

To evaluate the Core-Fixed model’s performance in different relevant scenarios, we will use a few different test sets.

Test-Original - This constitutes of all molecular pairs used for testing purpose, corresponding to a size of 19,856 pairs. With this construction we see that Test-Original will have the same molecular space as the training and validation sets. For this test set we will use the source molecules, as well as the property change tokens corresponding to the target molecules in the test set.

Test-Core - This is a subset of the molecular pairs in Test-Original where we have excluded molecular pairs for which the core is present in the training set, yielding a test size of 6,136. When working with the Core-Fixed model, the purpose of Test-Core will be to see how well the models generalize on unseen cores.

Test-Property - This set consists of 7,813 starting molecules with low Solubility, high Clearance, and LogD between 2 and 4.4 in Test-Original. For this test set we are interested in achieving lower LogD, high Solubility and low Clearance. The desirable LogD for the target molecules are constrained to lie in the interval 1.0 to 3.4, as the experimental data for which the internal property prediction model was trained on, lies in this range. The purpose of Test-Property is to evaluate the models on how well they perform on the particular property changes we are interested in. As was noted in the section of curriculum learning, this specific property change is

only used by 0.2% of the molecular pairs in the training data.

3.4.5 Evaluation Metrics

The evaluation of the Core-Fixed model and its baselines will be similar to what was presented for curriculum learning in Section 3.3.7. Besides those evaluation metrics, i.e. Desirable, Novel Transformations and MMP33, we will in the case of the Core-Fixed model also consider the following:

Novel R-groups - This metric gives the proportion of generated molecules that contain R-groups which are not seen among the R-groups in the training data. This metric will give an idea of models' generalization performance. As for Novel Transformations, the performance interpretation will depend on the corresponding Desirable value.

Unchanged Core - This refers to the proportion of generated molecules that keep the core specified by model input. This metric is an approach to verifying how well the original Transformer for molecular optimization (Baseline) compares to the Core-Fixed model and the Enumeration Baseline, since the corresponding proportions for these will be 100% by constructions. Here a higher value is preferred.

4

Results

This chapter goes through the results for the two extensions of the previous Transformer for molecular optimization; curriculum learning and Core-Fixed formulation.

4.1 Curriculum Learning

In the following sections the results for the curriculum learning Transformer models are presented. The chapter begins by comparing the different difficulty assessments that were discussed in Section 3.3.2. This will be done firstly by looking at the model’s performance and development during the training and Validation. After this, the Evaluation Metrics will also be used to compare them in terms of generated molecules. Furthermore, a comparison of the computational time will be done. Throughout the chapter, *Baseline* refers to the Transformer model trained without the use of curriculum learning as in [1].

4.1.1 Comparison of Exploratory Difficulty Assessments

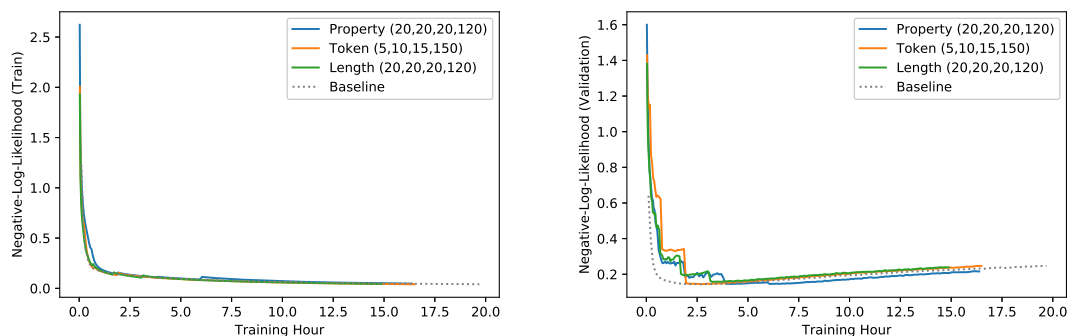
This section compares models trained using the different difficulty assessments. To make the results easier to interpret, they have been split up in a part covering the Exploratory difficulty assessments, which are our main results, and one covering a comparison of the Property-Based difficulty assessment with the Comparative difficulty assessments. This part covers the Exploratory difficulty assessments.

4.1.1.1 Training and Validation

In this section, the training and validation loss curves for the Exploratory difficulty assessments are shown. For the choice of number of epochs to train in each step for each model, a few different choices were tested and the best one of them was chosen. See Appendix A.1 for more information on this.

In Figure 4.1 we see that training loss is very similar for all models. Looking at the validation loss we see that the Exploratory models differ from the baseline in the early epochs, only to later sync up more. The reason for this could be that the Exploratory models only have access to easy subsets of the data during the early epochs, which makes them perform worse on the validation set, which contains both easy and difficult data. For all curves we see a slight upward tendency in the later part of the training, indicating that overfitting is occurring there.

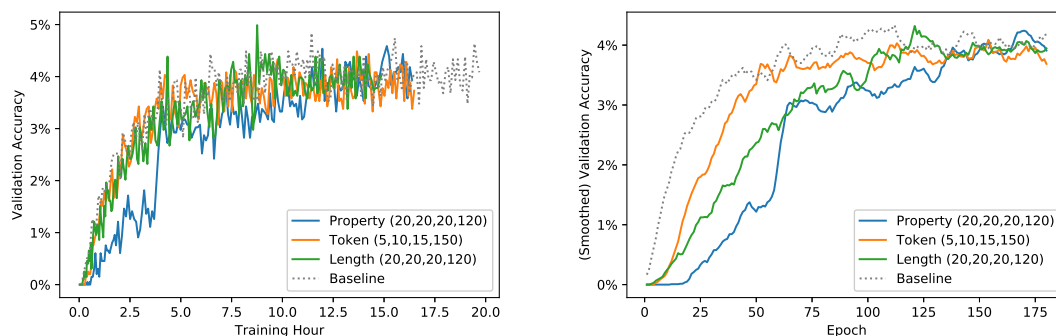
4. Results



(a) Training loss over training time

(b) Validation loss over training time

Figure 4.1: Training and validation loss over training time for the Exploratory difficulty assessments in curriculum learning. The numbers in the parenthesis represent the number of training epochs for each curriculum learning step.



(a) Validation accuracy over training time

(b) (Smoothed) validation accuracy over epoch

Figure 4.2: Validation accuracy over training time (a) and smoothed validation accuracy over epochs (b) for the Exploratory difficulty assessments in curriculum learning. The validation accuracy is defined in Section 3.3.5. The numbers in the parenthesis represent the number of training epochs for each curriculum learning step.

In Figure 4.2 we see the validation accuracy for the Exploratory difficulty assessments over (a) training time and (b) epochs. The validation accuracy that the model reaches can be seen as an indicator for how good the model will be at generating new molecules with desirable properties, which is the main goal of the model. In Figure 4.2(a) we see that there seems to be *no significant difference* in the training time of the models. If anything, Property is *slower* to converge than the rest, including the baseline. Similarly there seems to be *no significant difference* in what value the validation accuracy reaches, indicating that the models will perform the same on the test sets. In Figure 4.2(b) we see similar information, but we see it in terms of epochs instead. We see that the Exploratory models need more epochs to converge, which

is not surprising as previously mentioned, because the early epochs contain less data.

For **model selection**, i.e. the choice of epoch number to be used for evaluation on the test set, a trade-off between high *smoothed* validation accuracy and low validation loss, was considered. Here, the smoothed validation accuracy refers to the *running average* of the validation accuracy. Through this trade-off, combined with a subjective touch, the **epochs** were chosen to be the following:

- **Property:** 169
- **Token:** 153
- **Length:** 120
- **Baseline:** 111

4.1.1.2 Sample-Based Molecule Generation

In this section the Evaluation Metrics are given for the Exploratory difficulty assessments.

Table 4.1: Comparison of the generation performance of the Exploratory difficulty assessments (DA) and the corresponding baseline on the three test sets when using multinomial decode. See Section 3.3.7 for definitions of the evaluation metrics.

		DA	Test-Original*	Test-Molecule*	Test-Property*
Desirable	CL Transformer	Prop.	48.44%	48.42%	35.45%
	CL Transformer	Token	50.87%	50.29%	39.12%
	CL Transformer	Length	50.41%	51.55%	40.95%
	Baseline		51.81%	51.34%	37.82%
MMP33	CL Transformer	Prop.	89.42%	87.29%	89.21%
	CL Transformer	Token	90.40%	88.61%	89.83%
	CL Transformer	Length	89.71%	88.26%	90.22%
	Baseline		90.15%	88.28%	89.64%
Novel Trans.	CL Transformer	Prop.	51.18%	43.56%	57.31%
	CL Transformer	Token	49.18%	41.85%	58.28%
	CL Transformer	Length	50.90%	43.73%	58.36%
	Baseline		50.79%	43.11%	56.87%

In Table 4.1 we see the Evaluation Metrics for the Exploratory models. Looking at the Desirable metric, we see that Length performs the best, getting the highest percentage on Test-Molecule* and Test-Property*. In the same sense, Property performs the worst. The differences are small, however, and might not be significant.

Looking at MMP33, which gives the proportion of generated molecules for which the number of heavy atoms in the R-group is less than 0.33 of the number of heavy atoms in the entire molecule, we see that Token has the highest values on Test-Original* and Test-Molecule*, which means that it has learned to model the training data better. This difference is however also small, meaning it might not be significant.

For Novel Transformations, which is the proportion of generated molecules that yield a transformation (i.e. specific R-group change) which has not been seen in the training data, we see that Length gets the highest score on Test-Molecule* and Test-Property*, which means that Length generates many new molecules and therefore might be good at extrapolation. The differences are, however, small here as well.

For Test-Molecule*, we see that the models have similar Desirable and MMP33 values as to Test-Original*, which indicates that the models perform well on unseen molecules. The Novel Transformations is, however, lower on Test-Molecule*, which shows that the models use more known transformations on unseen molecules.

Finally, for Test-Property* the Desirable values are considerably lower than for the other two test-sets. Considering the low proportion of molecular pairs that have these associated property changes, it might not come as a surprise that the models score worse here.

4.1.2 Comparison of Property-Based with Comparative Difficulty Assessments

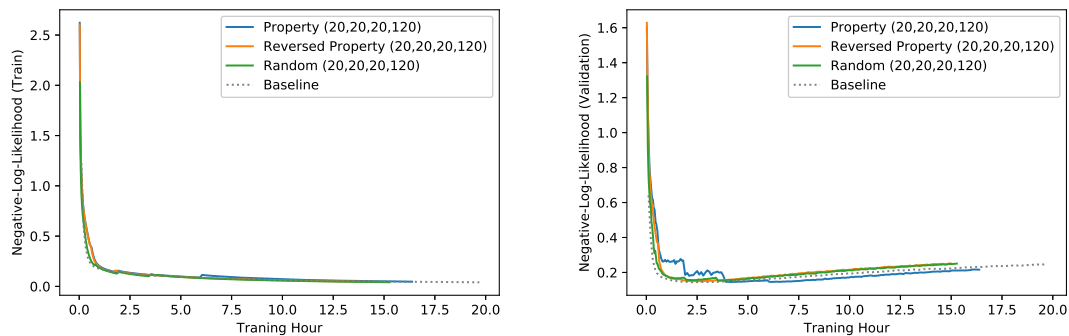
In this section we look at how the Property-based difficulty assessments compare to the two Comparative difficulty assessments Random-Based and Reverse (applied on Property-Based).

4.1.2.1 Training and validation

In this section, the training and validation loss curves for the Property-based difficulty assessment and for the Comparative difficulty assessments are shown. For the choice of number of epochs to train in each step for each model, a few different choices were tested and the best one of them was chosen, see Appendix A.1 for more information on that.

In Figure 4.3 we see similar curves as in the Exploratory case. We see again that training loss is very similar for all models and the validation loss for the Comparative models differ from the baseline in the early epochs. Again, the reason for this could be that the Comparative models only have access to easy data at that point. We also see the same slight upward tendency in the later part of the training, indicating overfitting.

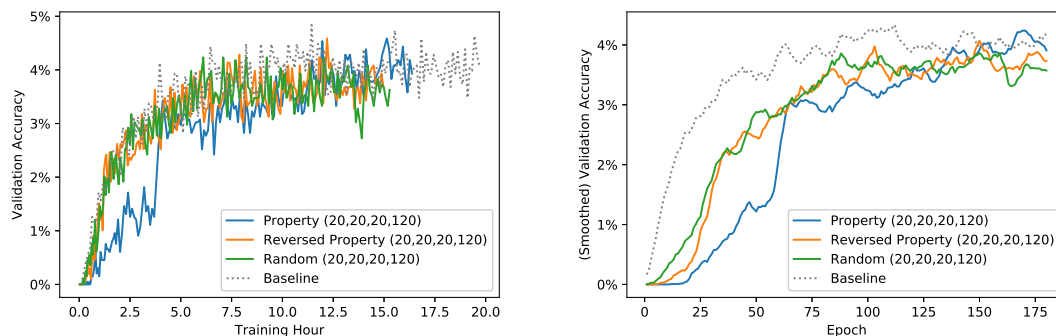
In Figure 4.4 we see the validation accuracy for the Comparative difficulty assessments over training time (a) and epochs (b). In Figure 4.4(a) we see, similarly to the Exploratory case, that there seems to be *no significant difference* in the training time of the models. If anything, Property is *slower* than the rest here as well. Similarly there seems to be *no significant difference* in what value the validation accuracy reaches, indicating that the models will perform the same on the test sets. In Figure 4.4(b) we see similar information, but we see it in terms of epochs instead. We see that the Comparative models need more epochs to converge, which



(a) Training loss over training time

(b) Validation loss over training time

Figure 4.3: Training and validation loss over training time for the Comparative difficulty assessments in curriculum learning. The numbers in the parenthesis represent the number of training epochs for each curriculum learning step.



(a) Validation accuracy over training time

(b) (Smoothed) validation accuracy over epoch

Figure 4.4: Validation accuracy over training time (a) and smoothed validation accuracy over epochs (b) for the Comparative difficulty assessments in curriculum learning. The validation accuracy is defined in Section 3.3.5. The numbers in the parenthesis represent the number of training epochs for each curriculum learning step.

is not surprising as previously mentioned, because the early epochs contain less data.

As with the Explorative case, the **model selection**, i.e. the choice of epoch number to be used for evaluation on the test set, a trade off between high *smoothed* validation accuracy and low validation loss, was considered. Through this trade off combined with a subjective touch, the **epochs** were chosen to be the following:

- **Random:** 158
- **Reverse Property:** 149

4.1.2.2 Sample-Based Molecule Generation

Table 4.2: Comparison of the generation performance of the Comparative difficulty assessments (DA) and the corresponding baseline on the three (masked) test sets. See Section 3.3.7 for definitions of the evaluation metrics.

		DA	Test-Original*	Test-Molecule*	Test-Property*
Desirable	CL Transformer	Prop.	48.44%	48.42%	35.45%
	CL Transformer	Rev. Prop.	50.77%	50.39%	37.01%
	CL Transformer	Random	49.23%	48.54%	37.20%
	Baseline		51.81%	51.34%	37.83%
MMP33	CL Transformer	Prop.	89.42%	87.29%	89.21%
	CL Transformer	Rev. Prop.	89.49%	87.35%	88.63%
	CL Transformer	Random	88.61%	86.44%	88.24%
	Baseline		90.15%	88.28%	89.64%
Novel Trans.	CL Transformer	Prop.	51.18%	43.56%	57.31%
	CL Transformer	Rev. Prop.	50.50%	42.63%	57.35%
	CL Transformer	Random	50.89%	43.34%	57.63%
	Baseline		50.79%	43.11%	56.87%

In Table 4.2 we see the Evaluation Metrics for the Comparative models. Looking at the Desirable metric we see that the baseline performs the best, getting a higher percentage on all of tree test sets. Furthermore, we note that the models using Reversed Property- and Random-based difficulty assessments get a higher percentage on all of the test sets compared to Property.

Next we identify MMP33, which gives the proportion of generated molecules for which the ratio between the number of heavy atoms in the R-group and the number of heavy atoms in the entire molecule is less than 0.33. We see that the baseline has the highest values on all of the three test sets, indicating that the curriculum learning models have not learned the data as good as the baseline has.

For Novel Transformations, we see that the Property-Based gets the highest percentage on two of the three test sets, which also seem to correlate with it having the lowest Desirable values.

4.1.3 Computational Time

In Figure 4.5, the training time is presented for each of the models in the previous sections. The training time is defined as the time required to get to the respective epoch for which model has converged according to the previously introduced criteria (model selection).

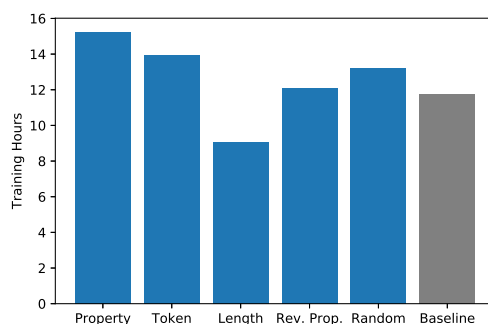


Figure 4.5: Training hours required for each of the Exploratory and Comparative models, including the baseline.

In Figure 4.5 it is seen that the training time vary significantly between the models. Most of the models are slower than the baseline, with only Length being faster. The Property based difficulty assessment required the most training time, while the Length based the least.

4.2 Core-Fixed Molecular Optimization

In the following sections the results for the Core-Fixed model are presented. The chapter begins with showing the model’s performance and development on the training and validation set. This is followed by results showing the molecule generation performance on the test sets. If not explicitly stated otherwise, the models use multinomial decode to generate 10 unique and valid molecules. Furthermore, a comparison of the computational time will be done. Through-out the chapter, *Baseline* refers to the Transformer model trained to generate molecules based on their entire SMILES representation, described in Section 2.2, as opposed to the Core-Fixed formulation.

4.2.1 Training and Validation

In Figure 4.6(a) the train and validation losses are presented over training epoch when using the same hyperparameters as for the original formulation [1].

It is seen that the model seems to suffer from overfitting which suggests looking for other hyperparameters to improve its performance. Appendix B.2 presents the corresponding losses for other hyperparameter choices.

In Figure 4.6(b) the change in validation accuracy over epochs is shown for the original Transformer model for molecular optimization and the Core-Fixed model with the same hyperparameters.

It is shown that the Core-Fixed model continues to improve in terms of validation accuracy until approximately epoch 38. Furthermore, Figure 4.6(b) also shows the Baseline (original Transformer model for molecular optimization) obtains a signifi-

4. Results

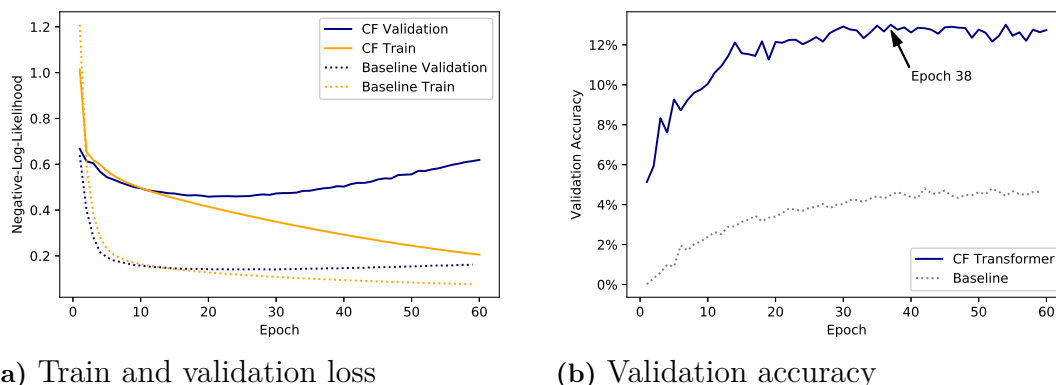


Figure 4.6: (a) Train and validation loss over epochs for the Core-Fixed model. (b) Validation accuracy over epochs for the original Transformer model for molecular optimization (Baseline) and the Core-Fixed model with the same hyperparameters. The validation accuracy is defined in Section 3.4.3.

cantly lower validation accuracy. In Appendix B.2 corresponding results are shown for other hyperparameter choices. Through a trade off between validation loss and validation accuracy the Core-Fixed model with the same hyperparameters as the original Transformer model for molecular optimization, was chosen for testing purposes. Using the same trade off between validation loss and validation accuracy, the number of epochs was chosen to be 38.

4.2.2 Deterministic Molecule Generation

When generating molecules during the validation procedure while training, it was seen that the Core-Fixed model yielded a significantly higher validation accuracy than its baseline. With this in mind, it is reasonable to look at how the same molecule generation procedure would act on the test data. Table 4.3 shows the proportion of generated molecules using greedy decode that fulfill the desirable properties along with the proportion of generated molecules that are the same as the corresponding target molecules.

Table 4.3: Comparison of the Core-Fixed model and its Baseline, when using greedy decode with one generated molecule per starting molecule, on Test-Original. **Top One Desirable** refers to the percentage of generated molecules that fulfill the desired properties while **Top One Accuracy** represents the percentage of generated molecules that are the same as their corresponding targets.

	Top One Desirable	Top One Accuracy
CF Transformer	70.19%	12.47%
Baseline	65.21%	5.00%

In Table 4.3 it is seen that the Core-Fixed model has a significantly higher ability to generate molecules with desirable properties. It is also seen that the proportion of

generated molecules that are the same as the true target molecule is approximately equal to the validation accuracies (see Figure 4.6(b)) which could be expected since Test-Original and the validation set are sampled similarly from the original data.

4.2.3 Sample-Based Molecule Generation

This section presents the main results for the Core-Fixed model and how it compares to its two baselines; the original Transformer for molecular optimization, and the Enumeration Baseline.

Table 4.4: Comparison of the performance of the Core-Fixed model and its corresponding baselines on the three test sets when using multinomial decode. See sections 3.3.7 and 3.4.5 for definitions of the evaluation metrics.

		Test-Original	Test-Core	Test-Property
Desirable	CF Transformer	58.97%	56.76%	42.90%
	Baseline	56.14%	55.61%	41.75%
	Enum. Baseline	16.93%	18.64 %	15.91%
MMP33	CF Transformer	97.67%	97.42%	97.57%
	Baseline	90.45%	86.82%	90.69%
	Enum. Baseline	77.85%	77.93%	81.19%
Novel Trans.	CF Transformer	53.92%	32.37%	57.84%
	Baseline	51.31%	34.76%	57.98%
	Enum. Baseline	96.62%	98.36%	96.65%
Novel R-groups	CF Transformer	4.30%	2.14%	4.66%
	Baseline	3.99%	2.27%	4.25%
	Enum. Baseline	0.00%	0.00%	0.00%
Unchanged Core	CF Transformer	100.00%	100.00%	100.00%
	Baseline	69.10%	44.60%	62.25%
	Enum. Baseline	100.00%	100.00%	100.00%

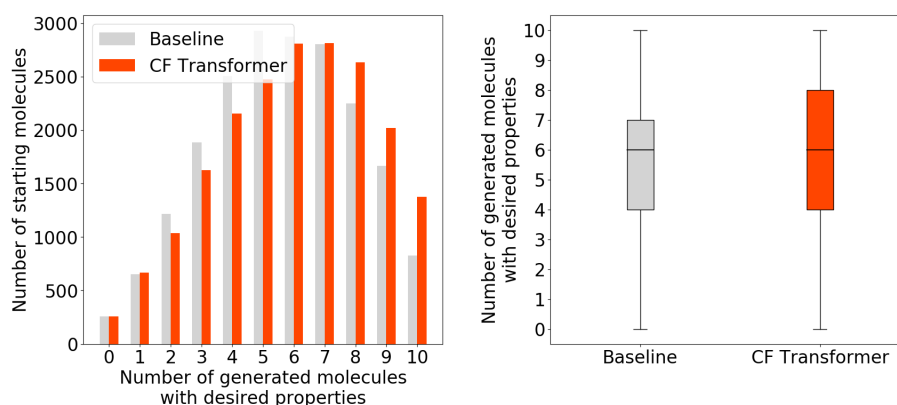
Table 4.4 presents various statistics of the generated molecules for the three considered test sets and the three models. Looking at the Desirable metric, it is seen that the Core-Fixed model manages to outperform the original Transformer for molecular optimization by 1-3 percent units depending on the test set. For the number of heavy atoms in the R-group compared to that of the entire molecule, seen through *MMP33*, Table 4.4 shows a clear advantage in using the Core-Fixed model.

For the proportion of generated molecules that obtain a transformation or R-group not seen in the training data (*Novel R-groups/Transformations*), the Core-Fixed model and the original Transformer for molecular optimization seem to obtain similar performances, with differences depending on the test set. In Table 4.4, it is shown that the two Transformer-based models yield lower proportion of generated R-groups which are not in the training data, for Test-Core than for the other two test sets. For the Enumeration Baseline no novel R-groups are generated, which is expected by design of the associated algorithm, while a high proportion of novel

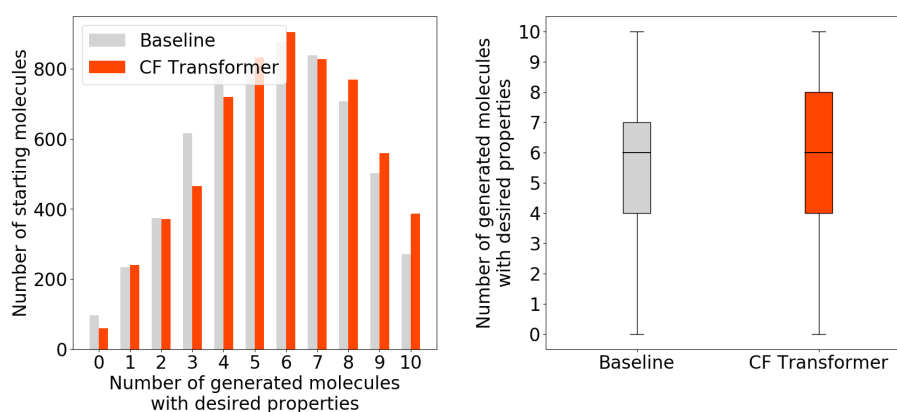
transformations are seen for the three test sets.

One of the main differences between the Core-Fixed model and the original Transformer for molecular optimization could be seen in Table 4.4 through the proportion of generated molecules which keep the core from the input molecules. For the Core-Fixed model and the Enumeration Baseline all generated molecules keep the core by constructions, while for the original Transformer for molecular optimization the corresponding proportion is considerably lower. This is in particular true for Test-Core containing input molecules for which the core has not been seen in the training data.

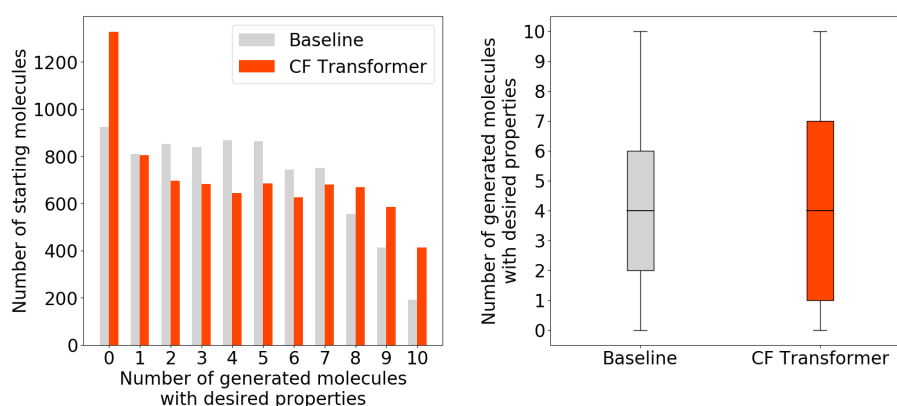
Finally, to get an idea of how the distributions of how many generated molecules that fulfill the desired properties for each starting molecule for the Transformer based models, we refer to Figure 4.7. As seen in Figure 4.7 the distributions are similar but with a slight advantage to the Core-Fixed model in terms of mean for Test-Original and Test-Core while the advantage on Test-Property is not as clear. Note that the differences in distributions seen in Figure 4.7 are statistically significant when using K-Sample Anderson Darling test (See Appendix C) at significance level 0.1%, but as the data sizes are big this itself might not serve as a strong indication of difference.



(a) (Left) Distributions of number of generated molecules with desirable properties per source molecule in Test-Original, (Right) corresponding box plots.



(b) (Left) Distributions of number of generated molecules with desirable properties per source molecule in Test-Core, (Right) corresponding box plots.



(c) (Left) Distributions of number of generated molecules with desirable properties per source molecule in Test-Property, (Right) corresponding box plots.

Figure 4.7: Number of generated molecules with desirable properties per source molecule when using the Core-Fixed model and its associated Baseline.

4.2.4 Novel R-Group Samples

Figure 4.8 shows the top 10 most common novel R-groups generated by the Core-Fixed model when generating based on Test-Original using multinomial decode.

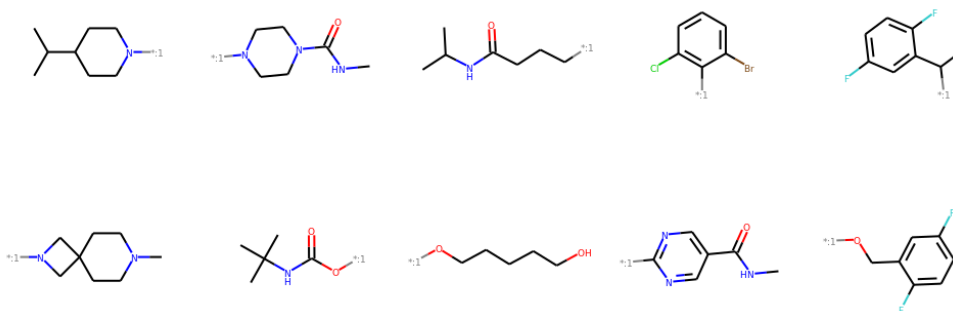


Figure 4.8: Top 10 most frequent novel R-groups found by the Core-Fixed model when generating using multinomial decode, on Test-Original.

4.2.5 Example of Baseline’s Inability to Keep the Core

As was seen in Table 4.4, a significant proportion of the generations from the Baseline failed to keep the core. In Figure 4.9 an example of such a source molecule is presented.

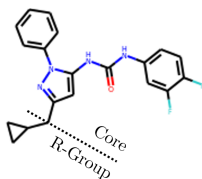
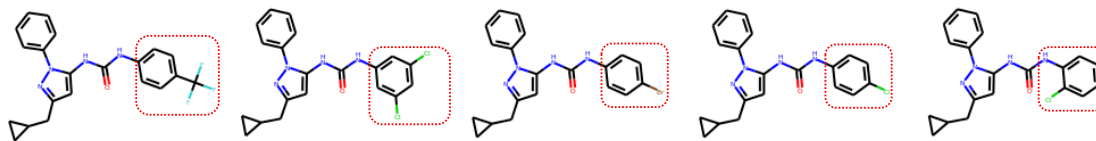
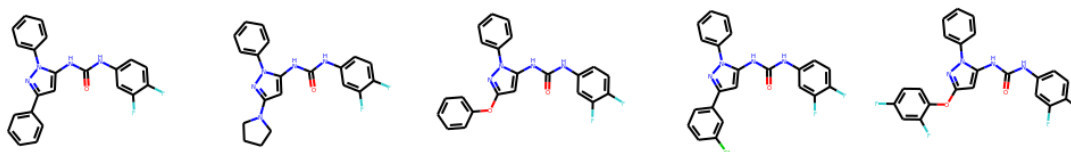


Figure 4.9: Example of a source molecule (CHEMBL2409118) for which the baseline failed to keep the core for the 10 generated molecules.

In Figure 4.10, five of the corresponding generated molecules using the Core-Fixed model and the Baseline respectively, are presented. It is seen that the Core-Fixed model succeeds in generating molecules where the core, seen in Figure 4.9, is kept. For the generations using the Baseline, however, it is seen that other parts of the source molecule are changed.



(a) Generations from the Baseline. The molecule parts surrounded by the red lines represent the erroneous transformations



(b) Generations from the Core-Fixed model

Figure 4.10: Generations based on the source molecule in Figure 4.9 for which the Baseline failed to keep the core. Note that all of the shown molecules fulfill the desirable properties LogD, Solubility and Clearance.

4.2.6 Computational Time

In Table 4.5 the training and generation times for the Core-Fixed model and its baselines are shown.

Table 4.5: Comparison of run-times for Core-Fixed model and corresponding baselines. For Generation, the Test-Original data set consisting of 19,856 molecule pairs, was used. Note that the generation time for the Enumeration Baseline is based on running the algorithm on 32 CPU:s. With more parallelization, a linear speed up in the number of CPU:s is theoretically achievable.

	Train	Generation
CF Transformer	7h 37m	7h 14m
Baseline	13h 8m	4h 47m
Enumeration Baseline	<i>(Not Applicable)</i>	32h 23m

It is seen that since the Core-Fixed model requires less training epochs, 38 as opposed to 60, the training time is decreased as compared to the original Transformer for molecular optimization. For generating molecules the Enumeration Baseline is slow and takes significantly longer time to finish generating molecules for Test-Original. In Table 4.5 it is seen that the generation time for the Core-Fixed model is slower than that of the original Transformer for molecular optimization.

5

Discussion

Following sections show a discussion among the results presented in the previous chapter and topics that can be explored outside the scope of this project.

5.1 Curriculum learning

In this section we discuss the results for curriculum learning concerning training and validation, and molecule generation. Based on these we suggest topics for future work.

5.1.1 Training and Validation

During training of the models, we saw that all of the models performed very similar, both in terms of training speed and in terms of validation accuracy. This suggests that curriculum learning, at least in the way it has been used in this project, does not significantly help our model. Furthermore, similar conclusions can be drawn from the Evaluation metrics, especially Desirable, which also showed similar values in all the models.

There are, however, slight differences in the different models, which are worth mentioning. When comparing the different difficulty assessments, we saw that Property had lower validation accuracy compared to the others in the lower epochs. A speculation for why this could be is that the curriculum arrangement for Property was made according to the available properties, and not by quartiles, making it have less data in step 1 than the other Exploratory models, as was seen in Figure 3.3. We also saw that, if any, the baseline and the Length-based difficulty assessment performed the best, both in terms of training time and in the Desirable evaluation metric.

Concerning the Comparative models: as stated earlier in the report, if curriculum learning would have helped in getting better results, in any way, we would expect to see that Property got good results, that Random got mediocre results and Reverse got bad results. What we saw in the validation accuracy was that there is no significant difference in the accuracy that they converge to, suggesting again that the use of curriculum learning is not very influential. If anything, we saw that both Random and Reverse actually had slightly *higher* validation accuracies than Property during the early epochs. A speculation for why this is could be that the easy data is *less informative* by having less properties, less tokens, or less informative tokens,

which means that the model might not learn as much from it. With this in mind, spending less time training on easy data, and more time training on difficult data, could improve the overall training time. This also suggests that the difficulty assessment was not beneficial to the model, although we guessed that it would. To make it work, other difficulty assessments should be tested, hopefully to find one that works.

The choice of the number of epochs to use for each curriculum learning step is both important and hard to decide. In this study a quite simple method of trial and error was used to find a good choice, but there could be better ones also. The choice is important when considering the training time, because for example if the first curriculum learning step includes more epochs than it takes for the baseline to converge, then curriculum learning will most likely not be able to get a lower training time. On the other hand, if the number of steps are too low, the model will be similar to the baseline, getting all the data (almost) at once. Similarly, the choice of convergence epoch is also important and hard to decide. It is important because it also affects the training time in a substantial way. A suggestion of how to choose the amount of epochs in each step is to make it more dynamical by evaluating (in some way) how much the model have learned in a step, and from that decide if it should move on to the next step or not.

5.1.2 Molecule Generation

When generating molecules, the Baseline seems to generally yield better performance than the curriculum learning alternatives. Interestingly enough, both Reversed Property- and Random difficulty assessments seem to outperform the other difficulty assessments. Again, this seems to show that what was first hypothesized as a good curriculum earlier in the report, might not actually be a good curriculum.

5.1.3 Future Work

With the above discussion in mind, a future work would be further hypertuning of the model parameters. This can include both tuning of the hyperparameters concerning the Transformer model, as well as various training parameters such as the definition of difficulty assessment. Besides the difficulty assessments considered within the scope of this project, another human-based difficulty assessment could consist of making a difficulty score based on the *average* token rarity in a SMILES input, as opposed to only the rarest — as was done in the token based difficulty. Another difficulty assessment, similar to the ones already tried out, is to do the assessment based on the similarity between the input- and output molecule, where a pair with similar molecules counts as an easy pair.

Previous works [19] have considered curriculum learning within NLP, for which it might be expected that humans have a greater understanding of what is considered hard, and thereby a better ability to design a curriculum, than in molecular optimization. Our results seem to suggest that designing a curriculum for molecular

optimization can be considered harder. For this reason an extension of this work might also be to instead try out machine-based difficulty assessments, as described in Section 2.4.1.

Investigation of more rigorous methods for choosing the amount of epochs in each step, and convergence epoch, as mentioned in the discussion, could also be helpful in getting more accurate evaluation.

5.2 Core-Fixed Molecular Optimization

In this section we discuss the results for the Core-Fixed model concerning training and validation, and molecule generation. Based on these we suggest topics for future work.

5.2.1 Training and Validation

During training, one of the observations was that the original Transformer for molecular optimization (Baseline) yielded a lower validation loss than the Core-Fixed model. With the model specific data inputs and outputs, this might be expected. Since the Baseline was trained on matched molecular pairs, the input and output only differ in terms of their R-groups. This means that input and outputs will be more similar than in the case of the Core-Fixed model that was trained on target R-groups which are not contained in the inputs.

When looking at the validation accuracy, we saw that the Core-Fixed model had a significantly higher validation accuracy than the original model. A potential idea to why this is could be that the Core-Fixed model is trained to model a smaller space, i.e. the space of R-groups' SMILES as opposed to the entire molecular space of SMILES. This can also be reflected by the training data for which the number of unique (target) R-groups are less than the unique number of target molecules.

5.2.2 Molecule Generation

During molecule generation it was seen that the Core-Fixed model outperformed the Baseline in multiple aspects. Partly the model generated more molecules with desirable properties, and partly the generated molecules consisted of R-groups which were more similar to those of the training data. The Core-Fixed model also managed to keep the core unchanged at a significantly higher rate than the original model, which in particular can be useful to chemists for whom a specification of which part of the molecule that should be changed might be necessary.

Furthermore, during molecule generation we saw that the Core-Fixed model required more time when generating molecules from Test-Original. This has most likely to do with the fact that the model requires further number of samples to obtain 10 unique and valid molecules that are not equal to the source, see Appendix B.1. What

this in turn could mean is that the Core-Fixed model generates less valid molecules (R-groups) or less unique molecules.

5.2.3 Future Work

It was seen that the Core-Fixed model yielded considerable improvement over the original model when using greedy decode. With slightly smaller improvements using multinomial decode in mind, future work could involve exploring the model potentials when using the more performance heavy beam search.

During training it was seen that Core-Fixed model seemed to suffer from overfitting. Through various hypertuning approaches, the severity of the overfitting was possible to reduce, however without an improvement in validation performance. To get around this issue, and to possibly increase the model performance, future work could consider some data augmentation. Since a molecule can be represented by multiple SMILES strings, one could for example consider enriching the data through using multiple representations of the same molecular pairs [35]. Another approach to reduce overfitting could be to increase the size of the data.

6

Conclusion

This thesis has covered two extensions of the Transformer for molecular optimization presented in [1]; (1) curriculum learning and (2) Core-Fixed formulation. Within molecular optimization, the goal is to improve promising molecules in terms of molecular properties which are important for drugs. Due to this usage, all models considered within the scope of this thesis have been evaluated based on how well the generated molecules satisfy the specified desirable properties.

The thesis covered an exploration of various human heuristic based curricula for curriculum learning within the context of molecular optimization. The final results indicated no clear differences to using these types of curriculum learning over a model trained on all data at once; neither in terms of training time nor performance. If any, the Length based difficulty assessment showed promising results and could be investigated further. This is, however, only one study, so no strong conclusions should be drawn.

The results suggest looking into more sophisticated machine based curriculum learning approaches. This was also motivated by the idea that manually designing a curriculum for molecular optimization could be seen as a harder task, than manually designing a curriculum for NLP where curriculum learning has shown success [19]. This could be due to the fact that humans have a more natural understanding of languages as opposed to molecules.

The second extension considered in the thesis, Core-Fixed formulation, was shown to yield significant improvement over the original model [1] in three aspects. Firstly, it was shown that the model managed to generate significantly more molecules with desirable properties. Secondly, the generated molecules were more similar to those of the training data, which suggests that the proposed model has a better ability to understand the molecular space. Thirdly, all generated molecules contain the desired part of the molecule (core) as opposed to the original model for which the corresponding proportion was significantly less. Since a generated molecule from the Core-Fixed model contains this specified part, the Core-Fixed model offers further potential for being used by chemists.

During training, the Core-Fixed model seemed to be sensitive to overfitting. The results suggest to increase the amount of data or to augment the data. This could be through enriching the data through taking usage of the fact that a molecule can be represented in multiple ways.

Bibliography

- [1] Jiazhen He, Huifang You, Emil Sandström, Eva Nittinger, Esben Jannik Bjerum, Christian Tyrchan, Werngard Czechtizky, and Ola Engkvist. Molecular Optimization by Capturing Chemist’s Intuition Using Deep Neural Networks. *Chemrxiv*, 9 2020.
- [2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.
- [3] John Noble Wilford. Lessons in iceman’s prehistoric medicine kit. *The New York Times*, Dec 1998.
- [4] Leland J. Gershell and Joshua H. Atkins. A brief history of novel drug discovery technologies. *Nature Reviews Drug Discovery*, 2(4):321–327, Apr 2003.
- [5] Christian Tyrchan and Emma Evertsson. Matched molecular pair analysis in short: Algorithms, applications and limitations. *Computational and Structural Biotechnology Journal*, 15:86 – 90, 2017.
- [6] Peter W. Kenny and Jens Sadowski. *Structure Modification in Chemical Databases*, chapter 11, pages 271–285. John Wiley and Sons, Ltd, 2005.
- [7] Andrew G. Leach, Huw D. Jones, David A. Cosgrove, Peter W. Kenny, Linette Ruston, Philip MacFaul, J. Matthew Wood, Nicola Colclough, and Brian Law. Matched molecular pairs as a guide in the optimization of pharmaceutical properties; a study of aqueous solubility, plasma protein binding and oral exposure. *Journal of Medicinal Chemistry*, 49(23):6672–6682, 2006. PMID: 17154498.
- [8] Ed Griffen, Andrew G. Leach, Graeme R. Robb, and Daniel J. Warner. Matched molecular pairs as a medicinal chemistry tool. *Journal of Medicinal Chemistry*, 54(22):7739–7750, 2011. PMID: 21936582.
- [9] Julia Weber, Janosch Achenbach, Daniel Moser, and Ewgenij Proschak. Vampire: A matched molecular pairs database for structure-based drug design and optimization. *Journal of Medicinal Chemistry*, 56(12):5203–5207, 2013. PMID: 23734609.
- [10] Pavel Polishchuk, Timur Madzhidov, and Alexandre Varnek. Estimation of the size of drug-like chemical space based on gdb-17 data. *Journal of computer-aided molecular design*, 27, 08 2013.
- [11] Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018. PMID: 29532027.
- [12] OpenSMILES. Opensmiles specification.

- [13] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning.
- [14] Jeffrey Elman. Learning and development in neural networks: the importance of starting small. *Cognition*, 48:71–99, 08 1993.
- [15] Guy Hachohen and Daphna Weinshall. On the power of curriculum learning in training deep networks, 2019.
- [16] Gaurav Kumar, George Foster, Colin Cherry, and Maxim Krikun. Reinforcement learning based curriculum optimization for neural machine translation, 2019.
- [17] Tabet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning, 2017.
- [18] Emmanouil Antonios Platanios, Otilia Stretcu, Graham Neubig, Barnabas Poczos, and Tom M. Mitchell. Competence-based curriculum learning for neural machine translation, 2019.
- [19] Benfeng Xu, Licheng Zhang, Zhendong Mao, Quan Wang, Hongtao Xie, and Yongdong Zhang. Curriculum learning for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6095–6104, Online, July 2020. Association for Computational Linguistics.
- [20] Kishore Papineni, Salim Roukos, Todd Ward, and Wei Jing Zhu. Bleu: a method for automatic evaluation of machine translation. *ACL*, 10 2002.
- [21] Neo Wu, Bradley Green, Xue Ben, and Shawn O’Banion. Deep transformer models for time series forecasting: The influenza prevalence case, 2020.
- [22] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.
- [23] Y. Kwon. Handbook of essential pharmacokinetics, pharmacodynamics and drug metabolism for industrial scientists. In *Springer US*, pages 44–45, 2002.
- [24] Stefan C. Kremer and John F. Kolen. *Field Guide to Dynamical Recurrent Networks*. Wiley-IEEE Press, 1st edition, 2001.
- [25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [26] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.
- [27] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.
- [28] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Hierarchical generation of molecular graphs using structural motifs, 2020.
- [29] Alexander Rush. The annotated transformer. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 52–60, Melbourne, Australia, July 2018. Association for Computational Linguistics.
- [30] A Gaulton, L. J Bellis, A. P Bento, J Chambers, M Davies, A Hersey, Y Light, S McGlinchey, D Michalovich, B Al-Lazikani, and J. P Overington. ChEMBL: a large-scale bioactivity database for drug discovery. *Nucleic acids research*, 40(D1):D1100–D1107, 2011.

- [31] Andrew Dalke, Jérôme Hert, and Christian Kramer. mmpdb: An open-source matched molecular pair platform for large multiproperty data sets. *Journal of Chemical Information and Modeling*, 58(5):902–910, 2018. PMID: 29770697.
- [32] M. Swain. Molvs: molecule validation and standardization, 2018.
- [33] John Cumming, Andrew Davis, Sorel Muresan, Markus Haeberlein, and Hongming Chen. Chemical predictive modelling to improve compound quality. *Nature reviews. Drug discovery*, 12:948–62, 11 2013.
- [34] Kevin Yang, Kyle Swanson, Wengong Jin, Connor Coley, Philipp Eiden, Hua Gao, Angel Guzman-Perez, Timothy Hopper, Brian Kelley, Miriam Mathea, Andrew Palmer, Volker Settels, Tommi Jaakkola, Klavs Jensen, and Regina Barzilay. Analyzing learned molecular representations for property prediction. *Journal of Chemical Information and Modeling*, 59(8):3370–3388, 2019. PMID: 31361484.
- [35] Dean Sumner, Jiazhen He, Amol Thakkar, Ola Engkvist, and Esben Jannik Bjerrum. Levenshtein augmentation improves performance of smiles based deep-learning synthesis prediction, Jun 2020.
- [36] F. W. Scholz and M. A. Stephens. K-sample anderson–darling tests. *Journal of the American Statistical Association*, 82(399):918–924, 1987.

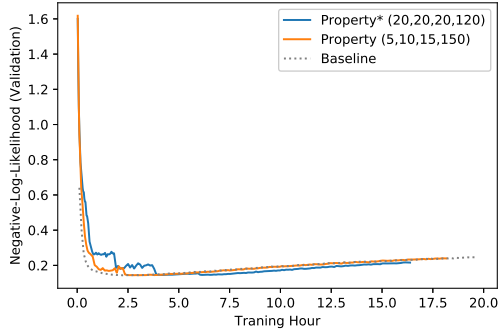
A

Additional Results for Curriculum Learning

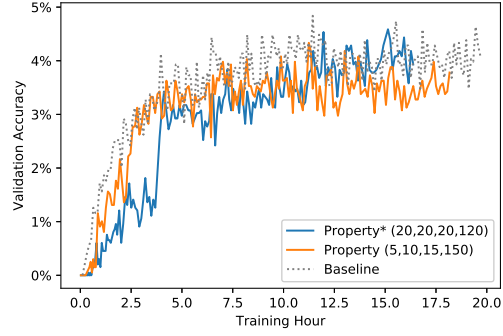
A.1 Choice of Epoch Numbers for Difficulty Assessments

A variety of different epoch choices was tried for the three difficulty appeasements. Each model were trained for a maximum of 180 epochs. In Figure A.1 the corresponding validation curves are presented.

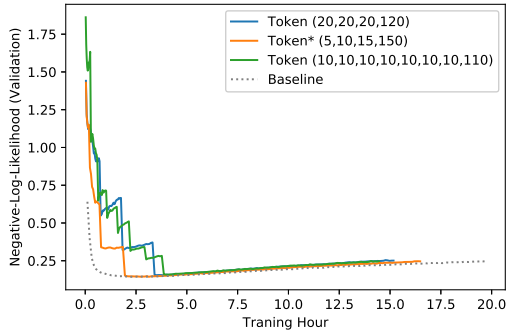
A. Additional Results for Curriculum Learning



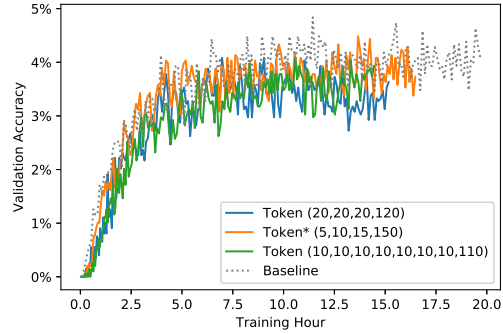
(a) Validation loss (Property)



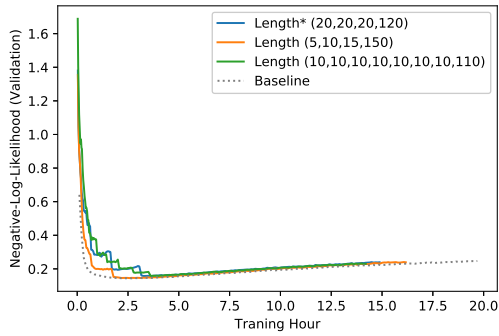
(b) Validation accuracy (Property)



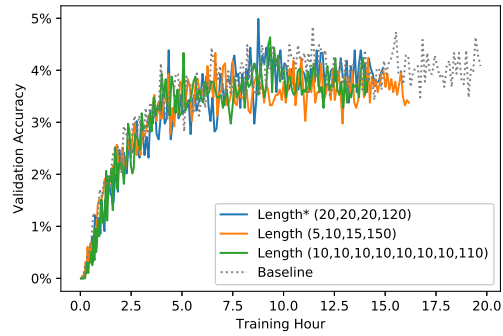
(c) Validation loss (Token)



(d) Validation accuracy (Token)



(e) Validation loss (Length)

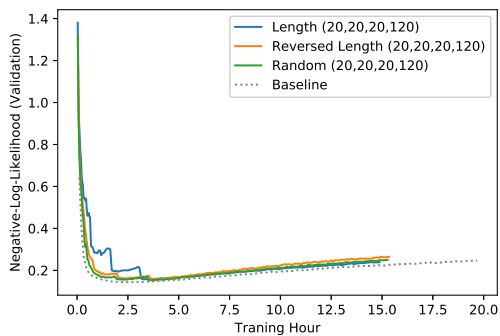


(f) Validation accuracy (Length)

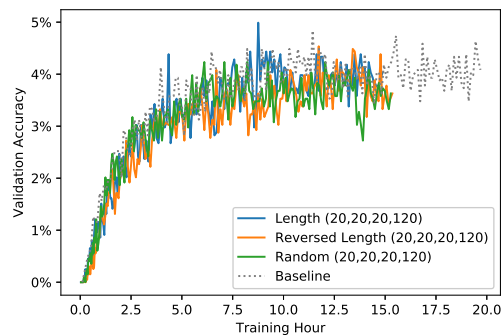
Figure A.1: Validation loss and validation accuracy for the three difficulty assessments. The models denoted by the star (*) denote the models used in Section 4.1. The numbers in the parenthesis represent the number of training epochs for each curriculum learning step.

A.2 Additional Comparative Results

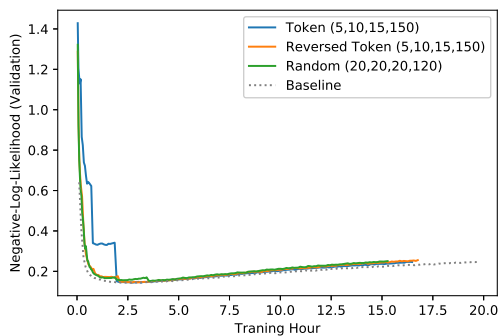
In Section 4.1 the Comparative results were presented based on the Property Based difficulty assessments. In Figure A.2 the corresponding training curves are presented for the Length- and Token based difficulty assessments. As for the Property based difficulty assessment, both the Token and Length based models perform slightly better than their reversed correspondent.



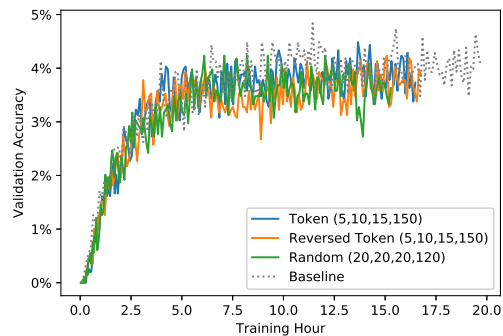
(a) Validation loss (Length)



(b) Validation accuracy (Length)



(c) Validation loss (Token)



(d) Validation accuracy (Token)

Figure A.2: Corresponding *Comparative* validation loss (a,c) and validation accuracy (b,d) for the length and token based difficulty assessments.

B

Additional Results for Core-Fixed Transformer

B.1 Required Number of Samples to Generate Molecules

In Figure B.1 we present the distributions of number of samples (out of 100) required to generate 10 unique and valid molecules for the Core-Fixed model and its Baseline, on Test-Original. We see that the Core-Fixed model seem to require more samples to obtain 10 unique and valid generated molecules.

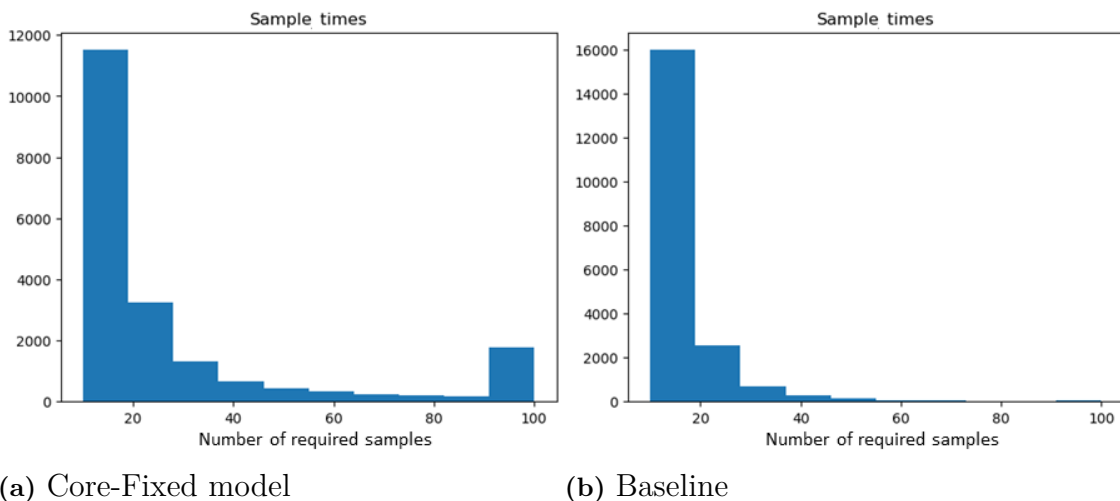
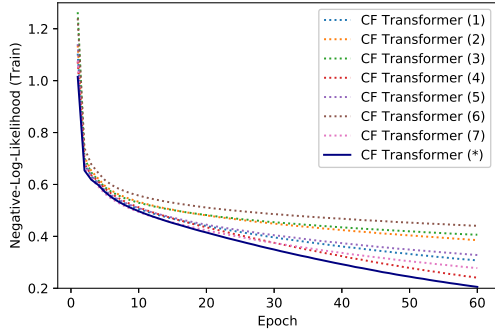


Figure B.1: Distribution of number of samples required to generate 10 unique and valid molecules for the Core-Fixed model and its Baseline.

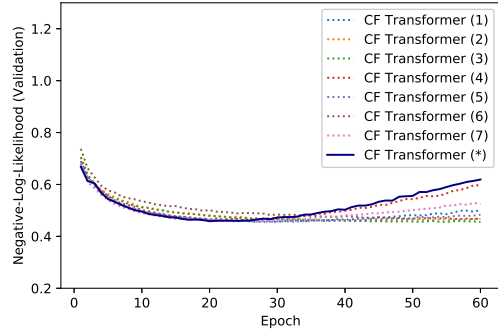
B.2 Hyperparameter Tuning

Tuning of the Core-Fixed Transformer model used the hyperparameters of the original Transformer for molecular optimization as a basis. Figure B.2 and Table B.1 present the training- and validation curves and specific hyperparameters corresponding to the 8 best hyperparameter combinations in terms of highest validation accuracy.

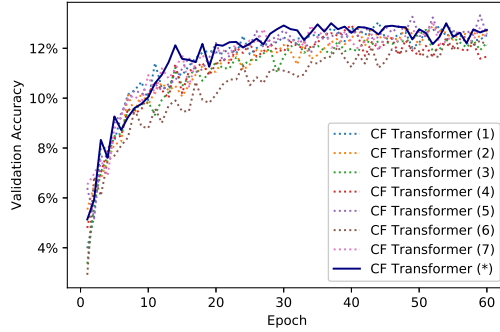
B. Additional Results for Core-Fixed Transformer



(a) Training loss



(b) Validation loss



(c) Validation accuracy

Figure B.2: Training loss (a), validation loss (b) and validation accuracy (c) over training epoch for the 8 hyperparameter combinations yielding the best performing models. The specific hyperparameter options are listed in Table B.1. The model denoted by the star (*) was the one chosen for testing in Section 4.2.

Table B.1: The 8 hyperparameters that yielded the best performing models. The model denoted by the star (*) was the one chosen for testing in Section 4.2.

Model\Hyperparameter	d_{model}	d_{ff}	N	h	Dropout
(1)	128	2048	4	6	0.1
(2)	128	2048	4	6	0.2
(3)	64	2048	4	6	0.1
(4)	256	2048	4	6	0.2
(5)	128	1024	8	6	0.1
(6)	128	2048	4	6	0.3
(7)	128	2048	4	6	0.1
(*)	256	2048	8	6	0.1

C

Testing for Significant Difference in Distributions

In order for comparing performance measures, we here introduce a framework to test for significance in distributions. This is relevant when wanting to compare the considered models in terms of successful generated molecules in a statistically proper way. This will be done using the so called "K-Sample Anderson-Darling test". This is a statistical test that yields a likelihood of whether k different samples are drawn from the same probability distribution or not, where the distribution itself is unknown [36].

The test can be described as follows. Let X_{ij} be the j^{th} observation in the i^{th} sample, $j = 1, \dots, n_i, i = 1, \dots, k$. where all the observations are independent. Suppose the i^{th} sample has distribution function F_i . Then, more formally, the null hypothesis that we want to test is

$$H_0 : F_1 = \dots = F_k$$

without specifying the common distribution F . Denote the empirical distribution function of the i^{th} sample by $F_{n_i}(x)$ and that of the pooled sample of all $N = n_1 + \dots + n_k$ observations by $H_N(x)$. The k -sample Anderson-Darling test statistic is then defined as

$$A_{kN}^2 = \sum_{i=1}^k n_i \int_{B_N} \frac{\{F_{n_i}(x) - H_N(x)\}^2}{H_N(x) \{1 - H_N(x)\}} dH_N(x)$$

where $B_N = \{x \in R : H_N(x) < 1\}$.

Under H_0 , this statistic follows a certain distribution, which you can use to when determining whether to support or reject H_0 .