CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Emerging Architectures for Chemical Language Modeling

Comparing Architectures for De Novo Molecular Generation

Master's thesis in Computer science and engineering

Ester Hagström
Erik Redmo Axelsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Emerging Architectures for Chemical Language Modeling

Comparing Architectures for
De Novo Molecular Generation

Ester Hagström

Erik Redmo Axelsson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Emerging Architectures for Chemical Language Modeling
Comparing Architectures for
De Novo Molecular Generation
Ester Hagström and Erik Redmo Axelsson

Supervisor: Daniel Brunnsåker, CSE
Advisor: Jiazhen He and Nicklas Österbacka, AstraZeneca
Examiner: Graham Kemp, CSE

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Emerging Architectures for Chemical Language Modeling
Comparing Architectures for
De Novo Molecular Generation
Ester Hagström and Erik Redmo Axelsson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

In recent years, language modeling architectures have become increasingly prominent in the field of generative chemistry, offering new approaches for the de novo design and optimization of small molecules. This thesis presents a comparative study of two emerging architectures: the decoder-only Transformer and the Mamba architecture, and a conventional Recurrent Neural Network with LSTM cells. The investigation explores how choices in training data, including a targeted medicinal dataset (ChEMBL) and a chemically broad dataset (PubChem), as well as data augmentation via randomized SMILES representations, influence generative capacity and chemical space coverage. In addition to this, task-specific optimization of models through reinforcement learning is studied, and the models are compared with respect to their ability to generate diverse molecules with desired properties.

Through pretraining experiments, it is shown that while the Mamba and RNN architectures reach their optimum performance significantly faster, the decoder-only Transformer achieves the highest validity and uniqueness in molecular generation. Training on PubChem, as opposed to ChEMBL, generally enhances validity and uniqueness but tends to reduce novelty, indicating a trade-off between chemical space saturation and innovation. As for data augmentation through randomization of SMILES, this helped all models refrain from memorizing the dataset, resulting in higher novelty across architectures and datasets.

Reinforcement learning experiments further reveal that all three architectures are capable of optimizing toward specific molecular properties, with the decoder-only Transformer and Mamba each exhibiting distinct strengths depending on the optimization task. Regarding the pretraining condition's effect on reinforcement learning, ChEMBL-trained models outperformed those trained with PubChem on multiple tasks, and all architectures, but especially Mamba, benefitted from being pretrained with randomized SMILES. Notably, even reduced-parameter models, such as a downsized decoder-only Transformer variant, perform competitively relative to larger architectures.

Keywords: Deep learning, chemistry, drug design, mamba, ssm, state space, transformer, LSTM.

Acknowledgements

We would like to express our deep gratitude to our supervisors at AstraZeneca: Jiazhen He and Nicklas Österbacka. Jiazhen, who, through her research in AI-driven molecule design, has laid the groundwork for this thesis to be possible. Her expertise and knowledge have been invaluable to this project. And Nicklas, who, through his unwavering support and guidance, has helped us overcome many difficult challenges and inspired us.

Our sincere thanks also go to our supervisor at Chalmers, Daniel Brunnsåker, for his valuable feedback, continuous guidance, and belief in us as students. His encouragement and constructive critique have helped us through every stage of this project.

Finally, we would like to express our gratitude to the Molecular AI team at AstraZeneca for their important feedback and valuable discussions. It really has been a pleasure working side by side.

Ester Hagström and Erik Redmo Axelsson, Gothenburg, 2025-09-18

Contents

List of Figures

List of Tables

1

Introduction

Developing a new drug is a complex task that conventionally relies on human expertise and iterative experimentation [1]. The process begins with a medicinal chemist or biologist identifying a biological target for a disease, such as a protein or gene. Once the target is chosen, experienced chemists employ their deep understanding of molecular structures and interactions to design and synthesize candidate molecules. This is not simply a matter of following textbook rules or rigid algorithms; rather, it requires both creativity and accumulated scientific knowledge. Once successfully synthesized, molecules are refined through multiple rounds of testing and optimization to enhance potency as well as safety. Despite advances in our understanding of biology and chemistry, this conventional, human-driven approach to drug design faces significant limitations. The process is not only slow and labor-intensive, but it also struggles to efficiently explore the immense number of possible drug-like molecules, which is estimated to be as large as 10^{23} to 10^{60} compounds [2]. While it can be worthwhile to recreate and incrementally modify known drugs, medical progress depends on the ability to design and synthesize *novel* molecules that can address unmet clinical needs. Despite the astronomical number of potential drug-like molecules, pinpointing one structure that perfectly balances novelty, efficacy, and safety is an immense challenge, akin to finding a needle in a haystack. Hence, balancing the many competing objectives in drug development, such as safety, absorption, and metabolism of a drug, often exceeds what can be systematically addressed by manual methods alone [3]. This is where data science and artificial intelligence (AI) offer transformative potential, reshaping the landscape of small-molecule drug discovery and development by complementing human expertise with advanced computational approaches.

The integration of AI technologies into the drug development pipeline offers the possibility of navigating vast chemical spaces, optimizing multi-objective criteria, and reducing the traditionally high costs and lengthy timelines associated with bringing new therapies to market [4]. Importantly, these data-driven approaches can amplify, rather than replace, human expertise by extracting actionable knowledge from increasingly complex and high-dimensional biological and chemical data.

Generative AI techniques for molecule design have been explored for years, with researchers employing diverse architectures such as recurrent neural networks, generative adversarial networks, and graph-based models to create novel chemical structures [5, 6, 7]. Building on this foundation, recent advances in large language models

(LLMs) and sequence-based deep learning architectures have inspired the development of chemical language modeling. Originally developed for natural language processing, these models have demonstrated remarkable effectiveness in representing, generating, and optimizing small molecules using textual representations [8, 9, 10]. By leveraging the flexibility and scalability of transformers and other autoregressive models, researchers can perform *de novo* design, generating novel chemical structures with tunable properties and rapidly exploring chemical space in a data-driven manner [11, 12]. While these models offer the ability to learn and represent vast chemical distributions, it is important to recognize that the underlying training data fundamentally defines the chemical space accessible to any language model. The breadth and diversity of this data set, as well as any biases it may encode, play a central role in shaping the generative capacity and practical utility of language model architectures for de novo molecular design.

A development in this area is the integration of reinforcement learning (RL) frameworks into molecular generation workflows [13, 14]. RL allows for the iterative optimization of generated molecules towards user-defined objectives, such as how well a molecule attaches to a biological target or how easy it is to produce. This is done by treating molecule generation as a goal-directed process, where a generative model learns to generate molecules for a specific task. This use of language model architectures together with RL enables the direct steering of molecular exploration, balancing the dual needs of novelty and optimality in candidate design.

In this work, a selection of emerging architectures for language modeling is investigated, focusing on how language modeling can be exploited to generate chemically valid and diverse molecules. By studying the contents and structure of the generated molecules, the architectures' ability to capture and encode the chemical space is also investigated. As a model's accessible chemical space is defined by its training data, this work compares two datasets: a smaller, curated set focused on drug discovery and medicinal chemistry, and a larger, more diverse set that includes substances beyond pharmacology and drug development. Further, the integration of reinforcement learning strategies to optimize models for specific drug discovery tasks, such as maximizing target activity and similarity to a query compound, is investigated. This study aims to provide a systematic evaluation of emerging language modeling architectures for molecular generation and their integration with reinforcement learning, with the goal of informing future development of AI-driven methods in early-stage drug discovery.

1.1 Background

Drug discovery is an interdisciplinary field that integrates knowledge from biology, chemistry, and computational sciences to design molecules that precisely interact with specific biological targets. Cheminformatics, positioned at the intersection of chemistry and computational science, offers computational tools to analyze, represent, and predict properties of molecules, accelerating the identification and optimization of candidates.

Drugs are made up of molecules, each formed by atoms arranged in precise patterns. The way these atoms are bonded together and the resulting three-dimensional structure of the molecule play a key role in determining how the drug interacts with its intended biological target [15]. Designing an effective drug requires meeting a range of specific criteria, and de novo molecular design focuses on creating entirely new chemical compounds that fulfill these requirements.

A cornerstone of modern cheminformatics and compound design is the existence of large, publicly accessible molecular databases. Among the most widely used are ChEMBL [16], which curates bioactive molecules with drug-like properties such as molecular weight and the presence of functional groups commonly seen in marketed drugs. For example, ChEMBL includes compounds with desirable absorption and stability profiles. Another key resource is PubChem [17], an extensive open chemistry database housing information on millions of compounds, their structures, properties, and biological activities. These resources support data-driven molecular design by offering diverse chemical structures and associated bioactivity data.

Another central aspect of cheminformatics and de novo design is the concept of molecular representation [18]. While it is common to see molecules represented as graphs, with nodes and edges corresponding to atoms and bonds, textual representations such as SMILES [19] are prevalent within cheminformatics. The details of how the SMILES representation encodes molecular structure is described in the theory Section 2.1. Because the SMILES representation depends on the order in which atoms are traversed in the molecular graph, a single molecule can be described by multiple, equally valid SMILES strings; generating these alternative forms by randomly varying the starting atom and the way that the molecular graph is traversed, known as randomized SMILES, is a useful strategy for data augmentation.

These string representations facilitate the application of sequence-based generative models, which generate molecules character-by-character as valid SMILES strings. Early work in this area utilized recurrent neural networks (RNNs) [20], a deep learning architecture with recurrent units that enables them to remember information from prior inputs. This allows them to capture dependencies across time, making them well-suited for modeling the sequential nature of SMILES strings. More recently, transformers [21, 22] have replaced RNNs for many tasks, and are used as the core architecture behind many LLMs today. Transformers make use of a self-attention mechanism to capture the relationship between different parts of a sequence. A newer deep learning architecture called Mamba [23] has started gaining interest because it uses a different method to efficiently handle and learn from sequences, making it especially fast during inference.

The REINVENT4 framework [14], developed at AstraZeneca, is a library of molecular design tools that aims to aid researchers in their design of small molecules. REINVENT4 utilizes an RNN architecture that has been pretrained on a large dataset of molecules represented as SMILES strings. This pretrained model, referred to as the prior, captures the underlying distribution of the SMILES in the training data, serving as the initial generator in the de novo molecule design process. To guide the design of molecules towards desired properties, REINVENT4 integrates RL for

goal-directed optimization. Within this RL framework, the user is able to define a scoring function, also known as an oracle, which quantitatively evaluates candidate molecules. The scoring function can encompass a wide spectrum of properties, such as predicted activity or physicochemical characteristics.

1.2 Objective and Research Questions

The primary objective of this study is to evaluate the capabilities and limitations of emerging language modeling architectures for the de novo generation and optimization of small molecules for drug discovery. By investigating how different language model architectures and the training data influence the diversity, validity, and chemical relevance of generated compounds, this work aims to provide insights that can inform the further development of AI-driven molecular design methods. In particular, this thesis seeks to explore the practical impact of training data selection on various architectures, as well as the architectures' susceptibility to reinforcement learning optimization. For this purpose, three different architectures will be studied: (1) The RNN, (2) The decoder-only Transformer, and (3) The Mamba architecture.

To address the overarching objective, the following specific research questions are posed:

- How do the RNN, the decoder-only Transformer, and the Mamba architecture perform in generating chemically diverse and valid small molecules?
- How does the choice of training data, comparing a curated medicinal focused set (ChEMBL) with a larger, chemically broader dataset (PubChem), affect the chemical space that can be generated and explored by these models? In addition to the two different datasets, the effect of augmenting the training data through randomized SMILES is studied.
- How do the different architectures respond to reinforcement learning for molecular optimization, in terms of generating diverse molecules with desired properties?

1.3 Limitations

The primary limitations of this thesis, due to constraints in time and prior knowledge, are the restricted set of architectures investigated. There exists a plethora of sequence-based architectures and variations thereof, many of which were beyond the scope of this study to explore. As a result, the conclusions drawn here may not generalize to architectures or approaches not included in the evaluation. Additionally, this work relies on specific training datasets, which inherently limit the chemical space accessible to the models and may introduce biases related to dataset composition and curation.

Further, the hyperparameter tuning has been limited, and hence the models employed in this thesis do not represent the optimal configuration of each respective

architecture. As a result, their comparative performance could be influenced by suboptimal parameter choices rather than intrinsic differences in architectural design. Optimal finetuning of each architecture separately could potentially lead to increased downstream task performance, but differ in impact between architectures.

Moreover, the evaluation of model performance and generated molecules has been carried out using in silico metrics such as chemical validity, diversity, and predicted activity. While being informative, these assessments may not fully capture real-world factors such as synthetic accessibility, biological efficacy, or safety. Experimental validation of the generated molecules was not performed, and thus the true applicability of these results to practical drug discovery remains uncertain.

All architectures, as well as training and RL regimes, were implemented as a part of the REINVENT4 framework. While this ensures consistency across experiments, it also means that findings are closely tied to the specific features and constraints of REINVENT4s implementation. Any unexamined limitations of the framework itself may therefore influence the observed performance and outcomes.

Lastly, the interpretability of the models and the rationale behind their choices have not been systematically explored in this work. As such, understanding of the decision-making process remains limited, which may pose challenges for model transparency and trustworthiness.

2

Theory

To use language model architectures for the generation of molecules, a textual representation of molecules is utilized, namely SMILES (Simplified Molecular Input Line Entry Specification) [19], which encodes molecular structures into strings. Using this textual representation, various language model architectures (such as RNNs [20], Transformers [21], or Mamba [23]) can be trained to generate molecules that reflect the data on which they have been trained. Once trained on a large set of SMILES, models can be focused on generating molecules with desired properties. This is done through reinforcement learning, through which a model is trained to maximize a score defined by the user. The process employ enhanced sampling, meaning that all molecules generated during reinforcement learning are collected. The theoretical details of this procedure will be described in the following sections, setting the stage for how these components come together in the methodology to answer the research questions of this thesis.

2.1 Simplified Molecular-Input Line-Entry System (SMILES)

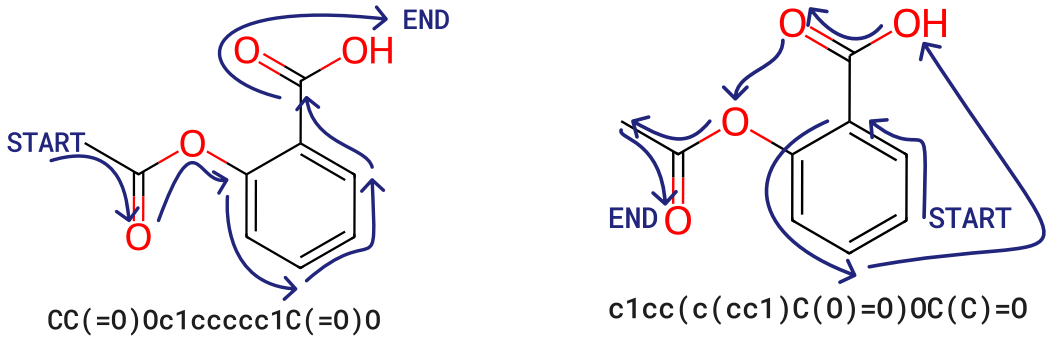
SMILES (Simplified Molecular Input Line Entry System) is a notation method that represents chemical structures using short ASCII strings. It is widely used for the input and storage of molecular information in computer-assisted chemistry as it allows for the compact representation of complex structures. In the SMILES syntax, atoms are represented by their atomic symbol, such as C for carbon and O for oxygen. Hydrogen atoms and most single bonds are usually implicit; hydrogens are inferred from each atom's standard valence, and single bonds are omitted from the notation, whereas there are special notations for double, triple and aromatic bonds. The SMILES syntax also includes syntax to describe properties such as aromaticity and stereochemistry.

As an example, Aspirin $C_9H_8O_4$, can be represented by the SMILES string:

$$CC(=O)Oc1ccccc1C(=O)O.$$

The SMILES string is constructed by traversing the molecular graph, making the SMILES path-dependent as the SMILES string is generated based on the order in which atoms are traversed through the molecular structure. Hence, for one compound, there exist many different SMILES representations as illustrated in Figure

2.1. To be able to identify if two SMILES strings represent the same compound, the strings can be canonicalized, i.e., converted into a unique, standardized form. However, this also means that a dataset can be augmented through randomization of a compound, where the first atom is chosen randomly and the molecular graph is traversed starting from this atom.



(a) Canonical path over the Aspirin molecule and corresponding SMILES string.

(b) Random path over the Aspirin molecule and corresponding SMILES string.

Figure 2.1: The Aspirin molecule, its canonicalized and randomized representations with starting nodes indicated by "START" in blue, and the path traversed to produce the SMILES string indicated by blue arrows. For the canonicalized representation, the starting node is chosen through a deterministic atom ranking algorithm, whereas for the randomized representation, the starting node is chosen at random.

2.1.1 Molecular Scaffolds

The molecular scaffold refers to the core structure of a molecule and characterizes its overall shape and connectivity. One such molecular scaffold is the Bemis-Murcko scaffold [24] (which we will refer to simply as Murcko scaffold). It is derived by removing side chain atoms, creating a reduced molecular graph. The scaffold can then be *genericized* into a generic scaffold, which is created by first identifying the Murcko scaffold of a molecule, and then genericizing it, meaning that all atoms in the scaffold are replaced with carbon atoms, and all bonds are converted to single bonds (other kinds of scaffolds exist and can also be genericized). The result is called a *generic Bemis-Murcko scaffold*, which further abstracts the molecule by focusing only on the topological framework. In Figure 2.2, the Murcko scaffold and the generic scaffold of a modified Aspirin molecule can be seen. The scaffolds make it possible to characterize groups of molecules based on their structure, providing a measure of variation in a larger group of molecules.

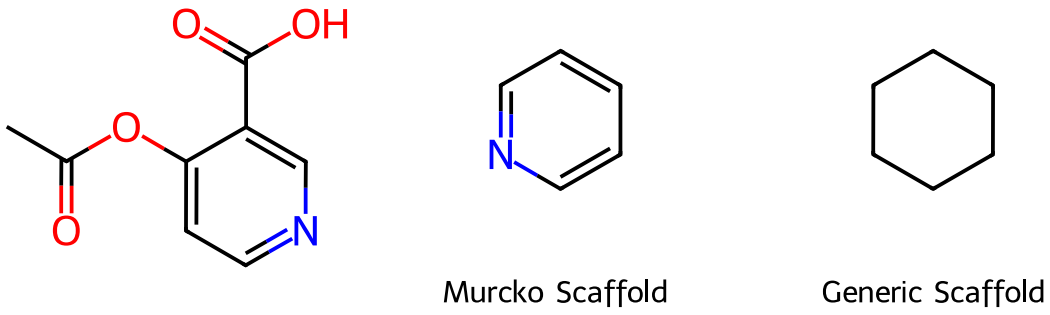


Figure 2.2: The Aspirin molecule with one of the carbon toms in the aromatic ring replaced by nitrogen, its Murcko scaffold, and its generic scaffold. The Murcko scaffold is derived by removing the two side chains of the Aspirin molecule, leaving only a hexagonal, aromatic ring. The generic scaffold is derived by replacing all atoms with Carbon atoms and all bonds with single bonds.

2.1.2 InChI and InChI Keys

Apart from the canonicalized SMILES format, a compound can also be identified using the International Chemical Identifier (InChI), which is a textual identifier for chemical substances. InChI representations are unique for every compound, meaning that non-canonicalized molecules and canonicalized molecules have the same InChI representation. Every InChI describes a compound in terms of layers of information, separated by "/". This InChI can be hashed to create a so called InChI key with a fixed character length of 27.

As an example, the Aspirin Molecule $C_9H_8O_4$, is identified in InChI format as

$$1S/C9H8O4/c1-6(10)13-8-5-3-2-4-7(8)9(11)12/h2-5H,1H3,(H,11,12),$$

with the corresponding InChI key

$$BSYNRYMUTXBXSQ-UHFFFAOYSA-N.$$

The use of InChI keys enables efficient and safe comparison between two molecules.

2.2 Tokenization and Embedding

To create a representation of the SMILES string that the generative model can learn, the SMILES strings are tokenized as illustrated by Figure 2.3. This is done by creating a vocabulary in which each distinct sequence of characters corresponding to a chemical component in SMILES is mapped to a unique integer. For example, the carbon atom, represented by `C` in the SMILES syntax, may be assigned the token `23`. In addition to tokens corresponding to the SMILES syntax, the vocabulary also holds a token to indicate the beginning of a new sequence, the beginning-of-sequence token (BOS), and a token to indicate when a sequence is finished, end-of-sequence token (EOS).

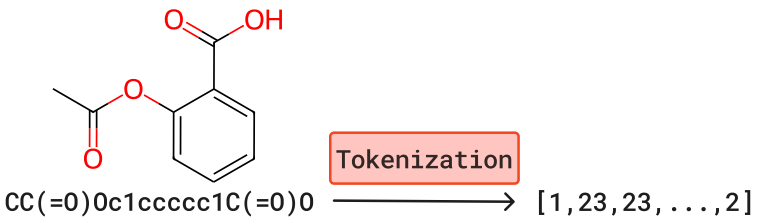


Figure 2.3: Tokenization of the Aspirin SMILES string. The SMILES string is tokenized, i.e., each character is mapped to an integer value using the vocabulary. In this vocabulary, the BOS token is represented by 1, and the EOS token is represented by 2.

After tokenization, an embedding layer maps these token representations of SMILES into continuous vectors. The weights in the embedding layer are learned features and make the model better understand the relationships and semantics between different tokens. This is a crucial part of how language models work. All models in this thesis operate on embedded tokens and not the tokens themselves.

2.3 Model Architectures

Deep learning architectures serve as the structural blueprints for how models process and learn from data. At its core, the architecture consists of multiple layers of connected neurons, and the unique ways to organize and connect these defines the various architectures from each other. In the context of chemical language modeling, with SMILES strings being encoded into sequences of tokens, the task requires models capable of processing sequential data. Throughout this section, we aim to explain the three different architectures that were considered in this thesis: (1) The Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) [25], (2) the Transformer architecture and its derivative: the decoder-only Transformer [21], and finally (3) State Space models in the Mamba architecture [23].

2.3.1 Recurrent Neural Networks

The Recurrent Neural Network (RNN) is a fundamental deep learning network architecture designed to process ordered sequential data such as time series, text, or speech [20]. By feeding an RNN sequential data as input, the network can learn to predict, classify, or generate the succeeding data point. To describe the order of the sequence, we refer to *time steps*, however, these time steps do not necessarily have to be in time but can be, for example, the order of characters in a string.

The RNN is made up of three types of layers: an input layer x, hidden layer(s) h, and an output layer o. Apart from connections from the input layer to the hidden layer(s), and hidden layer(s) to the output layer, the recurrent neural network

contains feedback connections from a hidden layer to the hidden layer itself. For visualization purposes, we can unfold the RNN, as seen in Figure 2.4. When unfolded, the sequential nature of the RNN is clearer as the input from time step $t-1$, x^{t-1}, is used to calculate the hidden layer output h^{t-1}, which is used to calculate the output o^t of the succeeding hidden layer h^t. In more detail, the output o^t is computed through

$$\begin{aligned} a^t &= b_1 + \mathbf{W}h^{t-1} + \mathbf{U}x^t \\ h^t &= \sigma(a^t) \\ o^t &= b_2 + \mathbf{V}h^t \end{aligned} \tag{2.1}$$

where b_1 and b_2 are biases, and $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ are weight matrices for the connections between the different layers. σ denotes the activation function, often set to a sigmoid function or ReLU. These weights are the primary learnable parameters of the network and are updated during training, usually via backpropagation through time. An RNN will use the entire sequence of data to predict, classify, or generate the output. In financial time series, an RNN can be used to predict tomorrow's stock price based on a previous time period. For text, an RNN can be used to generate the next word or character based on an input start sequence. One limitation of RNNs is that during training, for long input sequences, the influence of earlier data entries can become vanishingly small due to repeated transformations as information is propagated through many time steps. This is due to backpropogation using gradients to update the weights, and that these gradients can become vanishingly small in the computation of the gradient over many time steps; a problem known as the *vanishing gradient problem*. This makes it challenging for RNNs to retain long-range dependencies, since information from early in the sequence is compressed and can be "forgotten" by the time the last output is computed. This was partially solved by a subclass of RNN called Long Short-Term Memory (LSTM).

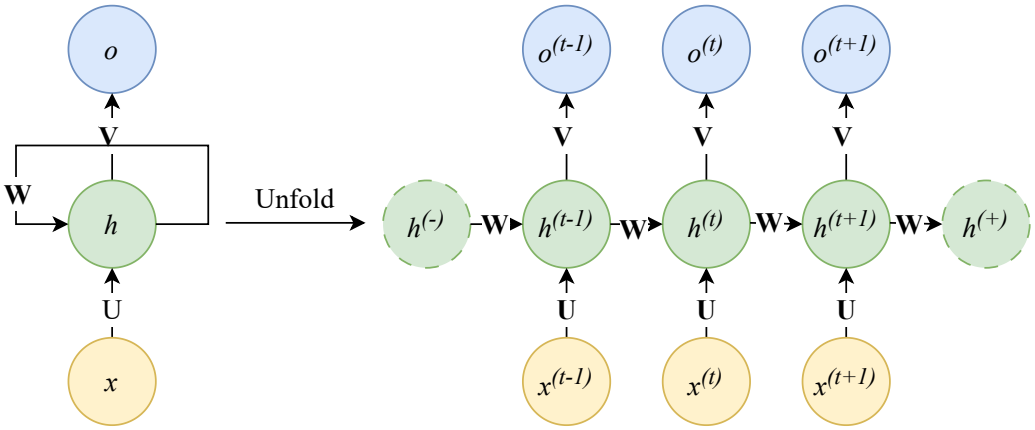


Figure 2.4: Schematic illustration of a Recurrent Neural Network (RNN) architecture. **U**, **V**, and **W** are the weight matrices that map the input x to the hidden layer h, the hidden layer h to the output layer o, and the hidden layer to the hidden layer itself, respectively. The left side depicts a single RNN unit, which processes an input at time step t, x^t, and produces a hidden state h^t by combining the current input with the previous hidden state h^{t-1}. The right side shows the corresponding unfolded RNN across multiple time steps. Each RNN unit shares parameters and passes its hidden state to the subsequent time step.

2.3.1.1 Long Short-Term Memory

One popular offspring of RNN is the Long Short-Term Memory (LSTM) [25]. This development of the RNN tackles the *vanishing gradient problem*. In other words, LSTMs are better at making the network "remember" information from longer sequences, or "periods of time". In the LSTM the hidden neurons of the RNN are replaced by computation units, so-called cells, that are designed to reduce the vanishing gradient problem.

The core of an LSTM cell is the *cell state*, which can be thought of as a memory line carrying information across many steps in a sequence, from cell state C_{t-1} to the next cell state C_t as visualized by the top most horizontal line in Figure 2.5. At each step, the LSTM decides how much information to add, remove, or keep in this memory. LSTMs use *gates* to control what happens to the information at each step. Each gate is made up of a neural network layer that uses a sigmoid activation function. This allows the gate to act like a filter or switch, letting some information pass through while blocking or discarding other information.

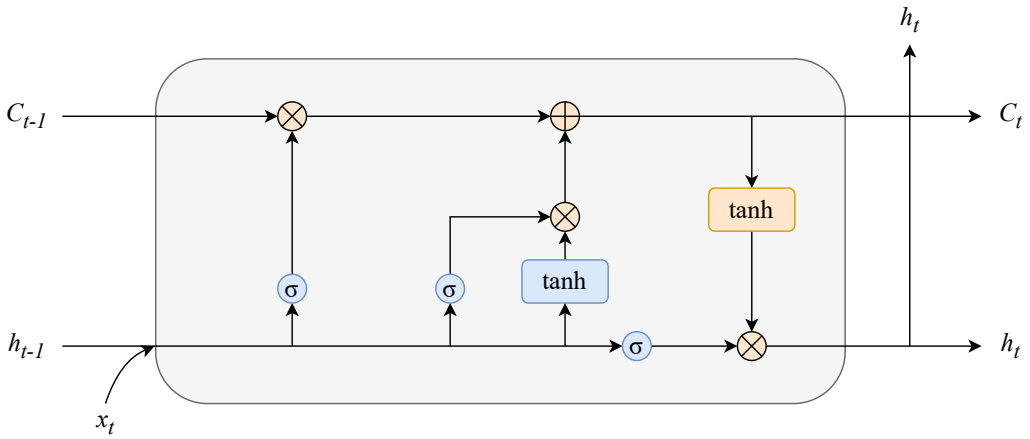


Figure 2.5: A schematic illustration of an LSTM cell. Within each cell, information flow is controlled by gates (forget, input, and output gates) marked with a σ followed by a multiplication $\otimes$. Into the cell goes (1) the input at the current time step x_t, (2) the hidden state from the previous time step h_{t-1}, and (3) the cell state from the previous time step C_{t-1}. From these, the hidden state h_t and the cell state C_t are computed and updated. Element-wise multiplication $\otimes$ and addition $\oplus$ operations combine and updates states within the cell.

There are three main gates in every LSTM cell: the forget gate, the input gate, and the output gate. The forget gate decides what information from the previous cell state should be thrown away. The input gate determines what new information should be added to the cell state. The output gate decides what part of the updated cell state should be sent as the LSTM's output for the current time step. At each step as a sequence flows through the LSTM, these gates work together to update the cell state. The LSTM network consists of a chain of cells, one for each data point in the sequence.

LSTMs were developed to address the vanishing gradient problem, allowing the network to retain and propagate information over longer sequences. In classic RNN and LSTM architectures, each input token is processed sequentially and the hidden state representation has a fixed size, independent of the input sequence length. However, this inherently sequential processing limits parallelization during training and inference. *Transformer* architectures overcome this limitation by processing all tokens in parallel using a *self-attention* mechanisms.

2.3.2 Transformers

As of 2025, many large language models (LLMs), including DeepMind's Gemini [26], Meta's LLaMA [27], and OpenAI's GPT model series [28, 29, 30, 31] build on the work introduced by Vaswani et al. [21]. Their proposed transformer architecture significantly advanced the performance of deep learning models on natural language processing tasks such as translation, summarization, and text generation, as seen in tools like OpenAI's ChatGPT.

The transformer relies on two key innovations: the attention mechanism and the encoder-decoder architecture. These developments emerged as solutions to core limitations in earlier sequential models such as RNNs and LSTMs, which struggled with long-range dependencies and parallelization.

2.3.2.1 Attention

The attention mechanism is based on the notion of selectively allowing models to become more influenced by the salient details and dynamically ignore less important information throughout the sequence. The mechanism was introduced by Bahdanau, Cho, and Bengio [32] in 2014 as a method to tackle the issue of using a fixed-length input and hidden state as demanded by RNNs. As RNNs compress all prior input information into a fixed-length hidden state, which causes information bottlenecks, especially for longer sequences. With the attention mechanism, however, each token is allowed to directly access all previous tokens in the sequence, without having to compress information. This removes the bottleneck of compressing all information into a fixed-length vector, making it easier to handle longer or more complex sequences. The attention mechanism was developed further by Vaswani et al., who introduced *scaled dot-product attention* and *transformers* [21] as an alternative to previous sequential network models.

The scaled dot-product attention (simply attention from hereon) maps the input sequence vector to three separate linear projections forming the key, query, and value vectors. Then, attention scores are generated using the key and query vectors of dimensions d. The attention scores are generated using the dot products of the query with all keys, divided by $\sqrt{d}$, and passed through the softmax function. This is done all at once for one sequence, as the full set of queries, keys, and values is packed into matrices $\mathbf{Q}, \mathbf{K}$, and $\mathbf{V}$. This is what makes training transformers parallelizable. The attention score is calculated as

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V}, \tag{2.2}$$

where softmax normalizes the input into a probability distribution through

$$\text{softmax}(\mathbf{z})_i = \exp(z_i) \,/\, \sum_j \exp(z_j)\,.$$

Following the attention mechanism, multi-head attention was introduced. Rather than relying on a single set of attention calculations, the queries, keys, and values are projected in h subspaces called "heads". While one of these computations is done on one head, *Multi-Head attention* performs the attention computation across multiple heads, h, in parallel as illustrated in Figure 2.6. Each head can learn different aspects of the data, as it allows the model to focus on different parts of the input sequence simultaneously.

Formally, for each of the h heads, the input matrices $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are linearly projected using parameter matrices $\mathbf{W}_i^{\mathbf{Q}}$, $\mathbf{W}_i^{\mathbf{K}}$, and $\mathbf{W}_i^{\mathbf{V}}$ (where i denotes the specific head), each with specific dimensions to split the internal representation across the

heads. Each head independently computes its attention output following Equation 2.2, but with these transformed projections. The individual attention outputs from all heads are then concatenated and projected once more with a parameter matrix $\mathbf{W}^O$ to form the final output of the multi-head attention block. This process is described by:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O \tag{2.3}$$

where each head is computed as

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^{\mathbf{Q}}, \mathbf{K}\mathbf{W}_i^{\mathbf{K}}, \mathbf{V}\mathbf{W}_i^{\mathbf{V}}). \tag{2.4}$$

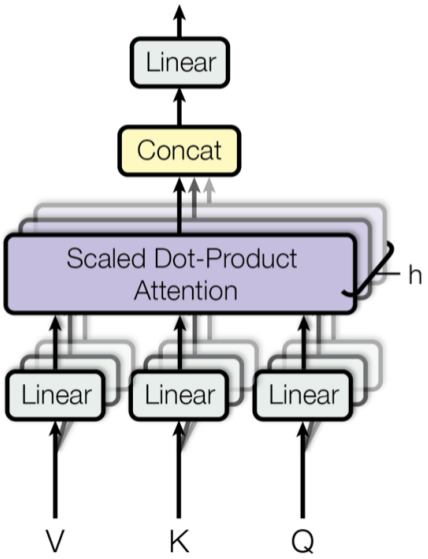


Figure 2.6: A visualization of multi-head attention [21]. "Attention Is All You Need - Multi-headed Attention" is licensed under CC BY-SA 4.0. Credit: Vaswani et al. [21], via Wikimedia Commons. The input consists of value $\mathbf{V}$, key $\mathbf{K}$, and query $\mathbf{Q}$ matrices, each first passed through separate linear transformations. Multiple sets of these matrices are fed in parallel into independent attention heads h, in which the scaled dot-product attention is computed. The output of the attention heads is concatenated and passed through a linear layer to produce the output.

Attention enables the model to establish direct relationships between each token embedding and all other token embeddings, regardless of their distance. When attention is applied within the same sequence, relating each token embedding to all others in that sequence, it is referred to as *self-attention*. In contrast, when attention is applied between two different sequences, relating elements from one sequence to elements of another, it is referred to as *cross-attention*. An extension of the self-attention is the so-called masked self-attention, which is used in the decoder layers of the transformer architecture. The masked self-attention is designed to ensure that token embeddings earlier in the sequence can not influence those succeeding them during training. This is done by *masking* the succeeding token embeddings before computing the attention scores.

2.3.2.2 Transformer Architectures

Vaswani et al. also introduced the transformer architecture, building on the development of attention. A visualization of the full transformer architecture can be seen in Figure 2.7. The architecture consists of two main components: the *encoder* and the *decoder* to the left and right of Figure 2.7, respectively.

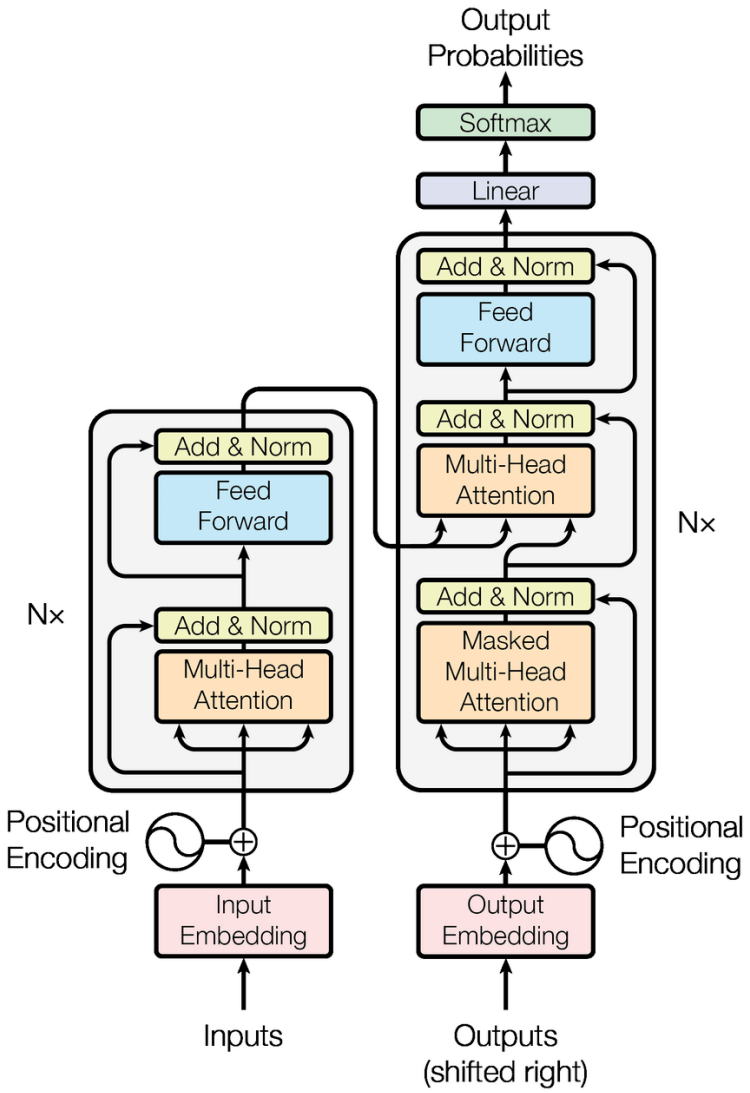


Figure 2.7: Schematic illustration of the Transformer architecture [21]. "Attention Is All You Need - Encoder-decoder Architecture" is licensed under CC BY-SA 4.0. Credit: Vaswani et al. [21], via Wikimedia Commons. The architecture consists of an encoder (left) and a decoder (right), each composed of a stack of identical layers. The encoder processes the input sequence into contextualized embeddings through layers of multi-head self-attention and position-wise feed-forward networks. The decoder, also constructed from stacked layers, receives these embeddings along with the autoregressively generated outputs, utilizing masked multi-head self-attention and cross-attention mechanisms to produce the output representations. These are then used to generate a sequence of output probabilities over the vocabulary.

The encoder comprises a stack of N identical layers, each consisting of two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. After each sub-layer, the input to that sub-layer is added to the output of the sub-layer through a residual connection.

The decoder stack, also composed of N layers, similarly consists of attention and feed-forward sublayers with residual connections. Different from the encoder layers, each layer has two different attention layers: the cross-attention layer, which attends to the output of the encoder, and the masked self-attention layer, in which a mask is applied to prevent tokens from attending to subsequent tokens during training.

Between each of the sublayers in both the encoder and decoder layers, normalization is applied. In the original Transformer architecture, the normalization is applied *after* each sublayer, commonly referred to as post-LN (LN for layer normalization). It has, however, been demonstrated that this way of normalization can make the training unstable [33]. By instead applying the normalization *before* each sublayer, so-called pre-LN, this can be improved.

While the original Transformer architecture uses both an encoder and a decoder, many modern models exclude the encoder component. For models where the task is to predict or generate the next token or generate sequences, such an encoder is often unnecessary. These decoder-only architectures have become the foundation of today's most prominent language models, focusing on autoregressive generation by predicting the next token in a sequence based only on an input sequence.

A decoder-only transformer model (decoder henceforth) comprises several transformer decoder layers with an identical structure that are stacked in sequence. In each of these blocks, the primary components are as described: the masked, multi-head attention and a feed-forward transformation with residual connections and layer normalization. The cross-attention mechanism for attending to the encoder's output is not needed in the decoder.

2.3.2.3 Positional Encoding

Unlike RNNs, Transformers do not rely on recurrence or sequential processing during training. Instead, they operate on the entire sequence simultaneously, enabling efficient parallel computation. On the other hand, this also means that there is no inherent sequence awareness in the architecture itself, which calls for the use of positional encoding to inject information about the position of tokens. This is what gives transformers the notion of long- and short-range context. This can be done using absolute positional encodings, which are added to the input embeddings, as seen in Figure 2.7. In the first version of the transformer architecture, sinusoidal signals with different frequencies were used as positional encoding by the addition of

$$PositionalEncoding(m, 2i) = \sin\left(\frac{m}{10000^{2i/d}}\right) \tag{2.5}$$

$$PositionalEncoding(m, 2i+1) = \cos\left(\frac{m}{10000^{2i/d}}\right), \tag{2.6}$$

to the token embeddings where m is the position of a token in the sequence, i is the dimension index, and d is the model dimension (output dimension of the embedding layer).

There are many other ways to inject this information, through learnable (optimized during training) and relative encodings (encodes the distance between tokens rather than their absolute positions). One way of positional encoding that uses absolute positions to construct relative positional encoding is Rotary Positional Encoding (RoPE), first proposed by Su et al. [34]. Instead of adding positional encodings to the embeddings, RoPE encodes positional information by rotating each pair of dimensions of the embedding using a sinusoidal function parameterized by the token position. This is done inside the self-attention mechanism by multiplying the query and key vectors by a rotation matrix, where the rotation angle depends on the token's position in the sequence. The rotation matrix takes the form of

$$\mathbf{R} = \begin{bmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & ... & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & ... & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & ... & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & ... & 0 & 0 \\ ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & 0 & ... & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & ... & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{bmatrix}, \quad (2.7)$$

where m is the position of a token and the angles θ_i are defined by

$$\Theta = \left\{ \theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, ..., d/2] \right\}. \quad (2.8)$$

The rotations are designed such that relative distances between tokens are preserved in the inner product (dot-product attention). This has been shown to make models generalize better to longer sequences than seen during training [34].

2.3.3 Mamba

The most common architecture for LLMs as of 2025 is built on the transformer architecture as stated in Section 2.3.2, but new ideas that might compete in efficacy or implementability are constantly being developed. At its core, the transformer architecture's self-attention mechanism and its ability to extract salient information within a finite context window are the underlying causes for its performance. It allows it to model complex data such as text or SMILES. However, the limited size of the context window in combination with autoregressive generation brings fundamental drawbacks: an inability to model anything outside of the finite window, and quadratic scaling of self-attention with respect to the sequence length. In the transformer architecture, self-attention computes interactions between all pairs of tokens, requiring $O(L^2)$ time and memory for a sequence of length L. This behavior is not prevalent in RNNs though, where the generation of the next token is only dependent on the previous hidden state and current input, as explained in Section 2.3.1. RNNs therefore scale *linearly* with the sequence length and can, in principle, process arbitrarily long sequences, though in practice their ability to retain and

utilize distant context is limited. Transformers face similar limitations if asked to extrapolate to contexts longer than those encountered during training.

In an attempt to merge the long-range dependency and parallelization of the transformer and the linear scaling and long context handling of the RNN, the *selective state space sequence models* (S6 or Mamba) architecture was proposed by Gu and Dao [23], built upon previous work by Gu et al. [35, 36] and Gu, Goel, and Ré [37] on *structured state space sequence models*. In the following subsections, the theory underlying the Mamba architecture will be explained, beginning with an introduction to state space models, progressing to structured and selective state space models, and culminating in a description of the Mamba block itself which is the essential part of the Mamba architecture.

2.3.3.1 Structured State Space Models S4

At time t, a state space model (SSM) maps an input sequence $x(t)$ to a latent state representation $h(t)$ and derives a predicted output sequence $y(t)$. Given a continous input sequence $x(t)$, the general state-space representation of a system can be described through

$$h'(t) = \mathbf{A}h(t) + \mathbf{B}x(t) \tag{2.9}$$
$$y(t) = \mathbf{C}h(t) + \mathbf{D}x(t), \tag{2.10}$$

where $h'(t)$ is the time derivative, i.e. change, of the hidden state $h(t)$ and $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ are learnable parameters. The system is visualized in Figure 2.8.

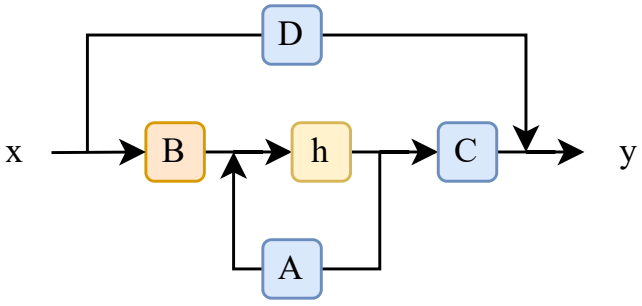


Figure 2.8: Schematic illustration of a State Space Model (SSM) architecture and Equations 2.9 and 2.10. The input and output signal is represented by x and y, respectively. Matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ are learned parameters in the SSM.

For tokenized SMILES strings, both the input and output sequences are discrete, requiring discretization of the continuous equations above. Any discretization can be used in principle, but the *zero-order hold rule* (ZOH) is commonly used for textual data. This method builds upon the idea that a discrete signal can be transformed into a continuous signal by holding each discrete value constant over a given step size Δ. Analogously, the same step size Δ can be used to translate a continuous signal to

discrete values. Other discretization methods, such as the bilinear transform, have also been applied [36]. A visualization of ZOH is shown in Figure 2.9. In practice, Δ is often treated as a learnable parameter.

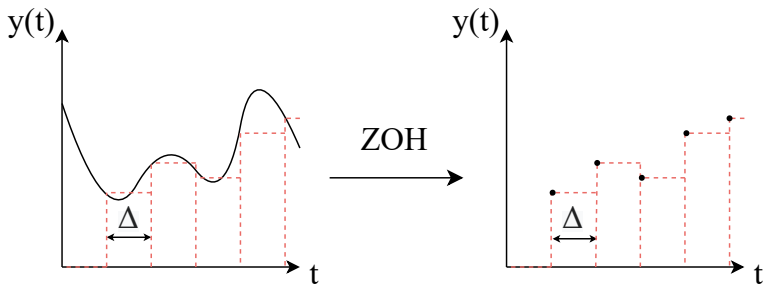


Figure 2.9: A visualization of the zero-order hold rule (ZOH), which transforms a continuous signal $y(t)$ into a discrete signal.

The ZOH rule can be applied to the continuous matrices $\mathbf{A}$ and $\mathbf{B}$ to create discrete representations $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$ through

$$\overline{\mathbf{A}} = \exp(\Delta\mathbf{A}) \tag{2.11}$$

$$\overline{\mathbf{B}} = (\Delta\mathbf{A})^{-1}(\exp(\Delta\mathbf{A}) - \mathbf{I}) \cdot \Delta\mathbf{B}, \tag{2.12}$$

where $\mathbf{I}$ is the identity matrix. Using the discrete parameters of the model, $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$, the discrete output y_t can be computed as

$$h_t = \overline{\mathbf{A}}h_{t-1} + \overline{\mathbf{B}}x_t \tag{2.13}$$

$$y_t = \mathbf{C}h_t. \tag{2.14}$$

Here, $\mathbf{D}$ is excluded as it acts as a skip connection outside the SSM block.

The discrete recurrent form above is natural for step-wise (autoregressive) inference, as one token can be predicted at a time with constant computation requirements per step. But, during training, all inputs and target tokens are accessible at once, making parallelization desirable. To facilitate this, the same system can also be represented in a convolutional form, where the output at each step is a convolution of the input sequence with a kernel. In linear, time-invariant systems, the recurrent and convolutional forms are equivalent; the recurrent form is ideal for inference and sequential generation, while the convolutional form enables fast, parallelized training on long sequences. The convolutional kernel $\overline{\mathbf{K}}$ is computed from the learned matrices and step size via $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$, as

$$\overline{\mathbf{K}} = \left(\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}\mathbf{B}}, ..., \mathbf{C}\overline{\mathbf{A}}^{k}\overline{\mathbf{B}}\right), \tag{2.15}$$

where k is the sequence length, the output can be calculated as

$$y = x * \overline{\mathbf{K}}, \tag{2.16}$$

where $*$ denotes the 1D discrete convolution operator, such that the output at each position is a weighted sum of input values over the kernel window. In Equations 2.13 and 2.14, $\overline{\mathbf{A}}$ influences the impact from the *state* of the previous timestep, while $\overline{\mathbf{B}}$ and $\mathbf{C}$ influence the impact from the current input value as well as the state of the current timestep respectively. The relation is very similar to that of an RNN as visualized in Figure 2.10. The recurrent nature of the SSM enables linear computational scalability and computationally supports the handling of sequences of arbitrary (infinite) length, as explained for RNNs in Section 2.3.1. The equivalence of the recurrent and convolutional views allows the SSM to be trained in parallel as a convolution, and run autoregressively and efficiently in inference as a recurrence.

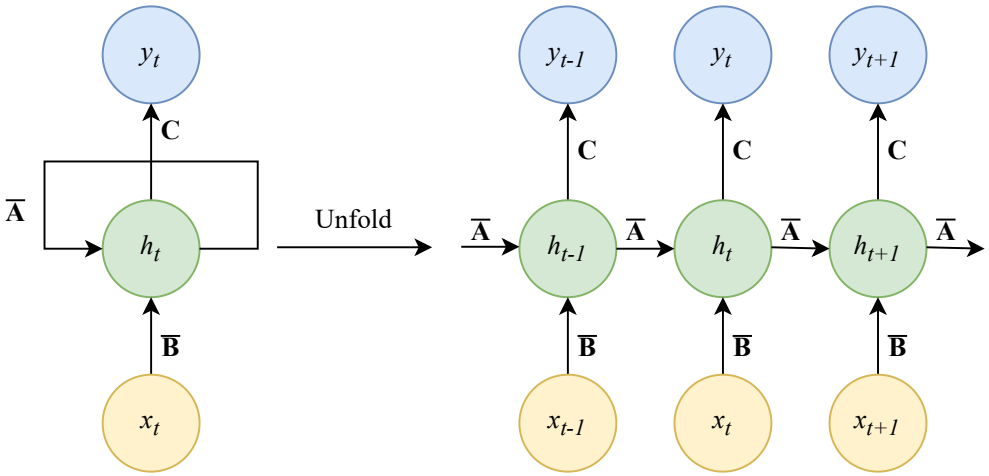


Figure 2.10: A visualization of the recurrent nature of the SSM. As the SSM is unfolded, the similarities with RNNs become clear. The matrix $\overline{\mathbf{A}}$ influences the impact from the state of the previous timestep, while $\overline{\mathbf{B}}$ and $\mathbf{C}$ influence the impact from the current input value as well as the state of the current timestep respectively. The calculations follow those of Equations 2.13 and 2.14.

The $\mathbf{A}$ matrix captures how information from the previous state, h_{t-1}, should be used to produce the new state, h_t. This is a central aspect of sequence modeling, and to ensure a well-behaved matrix $\mathbf{A}$, *high-order polynomial projection operators* (HiPPO) are utilized in the S4 model [35]. Inspired by Fourier transforms to decompose a signal, HiPPO utilizes Legendre Polynomials to build $\mathbf{A}$ in a way that it compresses all the input signals seen so far into a vector of Legendre polynomial coefficients. In this HiPPO representation of $\mathbf{A}$, its elements $\mathbf{A}_{n,k}$ take the form

$$\mathbf{A}_{nk} = \begin{cases} (2n+1)^{1/2}\,(2k+1)^{1/2} & n > k \quad \text{(below the diagonal)}, \\ n+1 & n = k \quad \text{(the diagonal)}, \\ 0 & n < k \quad \text{(above the diagonal)}. \end{cases} \tag{2.17}$$

The HiPPO representation is used as an initialization for matrix $\mathbf{A}$. This HiPPO-based structure enhances the model's ability to emphasize recent inputs more than

older ones and allows S4 to both retain information over very long sequences and be computed dynamically and efficiently during both training and inference.

While RNNs and transformers are each other's opposites when it comes to compressing the previous inputs (RNNs compress all history to the hidden state and the transformer operates on the full uncompressed sequence history), SSMs compress the input history using the HiPPO matrix to a partial degree. Combining state space models, the recurrent and convolutional techniques, and HiPPO, Gu et al. [36] and Gu, Goel, and Ré [37] created *structured state space sequence models* (S4), which is the main foundation for Mamba, alongside continued architectural innovations.

2.3.3.2 Selective SSM and Mamba

The structed SSM has a large context window, scales linearly in compute, and uses efficient memory handling via recurrent and convolutional representations and usage, plus HiPPO. It does however lack the ability to prioritize certain information over others because it is linearly time invariant, with the parameters $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ fixed for all sequence elements. Having shared matrices over all input tokens hinders the possibility of content-awareness for structured SSMs. In transformers, using attention as explained in 2.3.2.2 and 2.3.2.1, the models utilize the entire input sequence to compute the next token. This enhances content awareness but is computationally costly, while for S4, the compression using the HiPPO matrix initialization in 2.17 is computationally efficient but lacks content awareness. In Mamba, the compression of the input is instead selective, meaning it attempts to create content awareness by selectively incorporating input-dependent information.

The term *selective* in Mamba refers to the model's ability to adaptively determine, at each step, which information to prioritize, retain, or discard. This is achieved by dynamically generating the step size Δ and matrices $\mathbf{B}$ and $\mathbf{C}$ as linear functions of the current input, i.e., through learned linear projections or neural networks rather than using the same set of parameters for all incoming tokens. This dynamic parameterization enables the SSM to emulate content-awareness, allowing it to focus computation on the most relevant parts of the input, similar to the content-based selection provided by attention in transformers.

With the parameters being input-dependent, content awareness is enabled, but it also forces the parameters to be computed for each input, hindering linear scalability. This makes the convolution by kernel $\overline{\mathbf{K}}$ impossible, as it assumes time invariant parameters. To divert this quadratic complexity over the input length, Gu and Dao [23] uses *parallel associative scan* and an effective hardware-aware algorithm that, combined, enables linear-time scaling through parallel computing even though $\mathbf{B}$ and $\mathbf{C}$ are dynamically generated from the input at each timestep. The parallel associative scan operates by leveraging the associativity of the SSM update, i.e., it does not matter in which order intermediate states are combined, so that the dynamic recurrence can be computed efficiently for all positions in the sequence in one pass. This maintains the linear-time scaling even as per-token parameters change. The details on the parallel associative scan operation and the hardware-aware algorithm can be found in the original Mamba paper by Gu and Dao [23].

In the Mamba architecture, selective SSMs are used as partial blocks of a larger Mamba block, as the one seen in Figure 2.11. Inside the Mamba block, two projections are created through two separate linear layers. The first projection is passed through a convolution, followed by the SiLU activation function. The output of the SiLU activation is then passed through a dynamic SSM kernel which operates according to Equation 2.16. The kernel itself is constructed as in Equation 2.15, with the learnable parameters $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$ obtained by the ZOH rule, as described in Equations 2.13 through 2.14. The second projection also passes through a SiLU activation, before being multiplied by the output of the SSM kernel, which is projected down to the dimension of the input embedding through a linear layer.

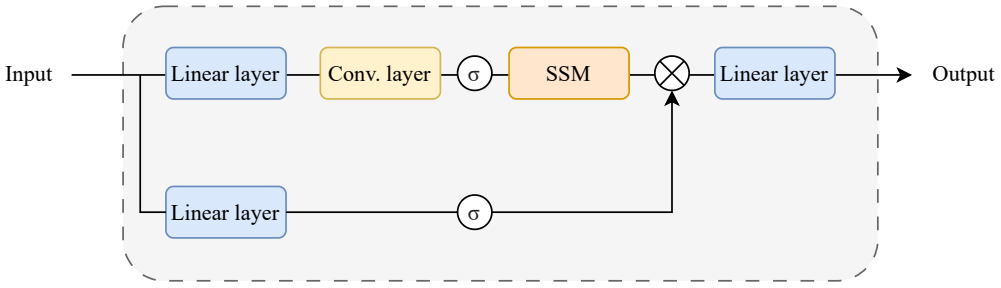


Figure 2.11: A visualization of the Mamba block. The input undergoes two separate linear projections; one branch passes through a convolutional layer, a nonlinear activation σ, and a selective SSM before a final linear transformation, while the other passes through a linear layer and activation σ. The two branches are gated by element-wise multiplication $\otimes$, followed by addition of a skip connection, and a final linear transformation generates the output of the block.

The full Mamba architecture consists of multiple stacked Mamba blocks, each one encapsulated by a residual connection and RMS norm. The final Mamba block is followed by an RMS normalization layer, a linear layer mapping the signal from the embedding dimension to the size of the vocabulary. Finally, the output passes through a softmax function to produce output probabilities over the vocabulary.

2.4 Learning the Data to Generate Molecules

In training generative models for the generation of SMILES sequences, the choice of loss function determines how the model parameters are updated to approximate the training data distribution. Given an input sequence of length $l-1$, taking the softmax function on the "raw" output of the model (logits) generates a probability distribution over the vocabulary, indicating what the l:th token of the sequence should be. Hence, to learn the underlying distribution of the training data corresponds to maximizing the likelihood that the token assigned the largest probability by the model is in fact the next token of that sequence. We can describe the probability of a sequence T of length l as the joint likelihood $\mathbf{P}(T|\theta)$ conditional on the

parameters of the model, θ, given by

$$\mathbf{P}(T|\theta) = \prod_{i=1}^{l} \mathbf{P}(t_i|t_{i-1}, t_{i-2}, ..., t_1; \theta), \tag{2.18}$$

where t_i denotes the i:th token of sequence T.

Motivated by this, the negative log-likelihood (NLL) is used as an objective to train generative models. For each sequence of tokens T, the NLL is defined as

$$\text{NLL}(T|\theta) = -\sum_{i=1}^{l} \log \mathbf{P}(t_i|t_{i-1}, t_{i-2}, ..., t_1; \theta). \tag{2.19}$$

During training, sequences are fed to the model, which for each position in that sequence produces a vector of logits representing unnormalized probabilities for each possible next token in the vocabulary. Through normalization and logarithmization of the logits, the model's predicted log-likelihood is quantified. The loss is then computed by comparing these predicted probabilities to the actual next tokens seen in the training data.

When training models in practice, it is common to group the sequences into batches. These batches are processed by the model simultaneously. The loss is then computed as the mean over the NLLs for the sequences in the batch, and the parameters of the model are updated using an optimization algorithm, once per batch. Over successive steps, the parameters of the models are updated to minimize the loss and find an approximation of the underlying distribution of the training data. For this thesis, *Adaptive Moment Estimation* (also called the Adam optimizer) [38] has been used to optimize the models.

A trained model will output logits that can be turned into a probability distribution over the vocabulary using the softmax function, given an input sequence of tokens. To use this to generate new sequences, from scratch, the model is initially fed a single beginning-of-sequence token (BOS), for which the probability distribution over the next token is output by the model. If simply choosing the next token as the one with the highest probability, so called *greedy* sampling, the same sequence of tokens would be generated over and over again. To address this, there exist multiple different sampling strategies, for example, multinomial sampling. Rather than always selecting the most probable token, the next token is sampled stochastically with a probability proportional to the output probabilities of the model. Once a token has been chosen, it is concatenated with the BOS token and fed into the model to produce new output probabilities. This process is repeated until an end-of-sequence token (EOS) is sampled, and the generation stops.

2.5 Reinforcement Learning for Task Optimization

The goal of pretraining the generative model is for the model to learn the chemical space (the probability distribution of possible token sequences) over which it has

been trained, making it capable of generating a variety of molecules representative of that specific distribution of compounds. In order to optimize the generative model to generate molecules with specific properties, such as activity towards a certain target or similarity with some other compound, reinforcement learning can be applied.

In reinforcement learning, an agent (typically a pre-trained neural network) chooses actions that maximize some reward signal given a state of a dynamical environment [39]. The set of possible states can be denoted as S, and the set of possible actions in a state $s \in S$ as $A(s)$. When choosing an action $a \in A(s)$, the agent receives a reward R and the agent's sole objective is to maximize the total reward over the course of the training. What actions the agent chooses at a given time and state s is defined by its policy π.

For the case of SMILES generation, the problem can be modeled as a Markov Decision Process (MDP), meaning that the next state and reward depend only on the current state and action, not on the history of prior states or actions. An action a corresponds to adding one token to the SMILES representation of a molecule, and the current state s is defined as the sequence of performed actions up until now, i.e, the sequence of tokens itself [40, 41]. Hence, the transitional probabilities of the MDP represent the probability of choosing a token a given a sequence s of tokens. If the immediate reward of choosing action a at state s is denoted as $r_t(a|s)$, the long term return can be denoted as $G(a_t, S_t = \sum_t^T r_t)$, that is, the cumulative reward up to some end point T. For the generation of SMILES, this endpoint T corresponds to when an EOS token is generated. The purpose of reinforcement learning is now for the agent to learn a policy π which maximizes the expected return $E[G]$.

Depending on the problem formulation, learning the policy π can look different. π defines the probability of choosing each possible action at a given state. In SMILES generation, it specifies the likelihood of selecting each token given the current sequence, guiding the agent's decisions to maximize the expected total reward. The optimal policy is stochastic and not deterministic, as the goal is not to generate the same compound over and over again but rather to generate a diverse set of compounds that all share some desired property. Therefore, the policy π should be a probability distribution over the possible actions a_t given a state s_t that maximizes the total reward. Meanwhile, the agent should also remember what has been learned during pre-training. For this purpose, a *prior* is defined, which is the pre-trained generative model [40], and the likelihood that this prior generates a SMILES is defined through a sequence of actions $A = a_1, a_2, ..., a_T$ as the product of the action probabilities

$$P(A)_{prior} = \prod_{t=1}^{T} \pi_{prior}(a_t|s_t). \tag{2.20}$$

The agent is then initialized as a copy of the prior, with the same probability distribution. To rate whether a sequence is desirable, let $S(A) \in [-1, 1]$ be a scoring function that returns a higher score for more desirable sequences, and a lower score for less desirable sequences (for examples of how this scoring function can be defined, see Section 3.4.1). To maximize the expected score for generated sequences, while

remembering what the prior has learned about the SMILES syntax and the chemical space of the training data, an augmented likelihood $\log P(A)_U$ is defined as

$$\log P(A)_U = \log P(A)_{prior} + \sigma S(A), \tag{2.21}$$

where σ is a scalar coefficient that scales the score $S(A)$ [42]. The loss $G(A)$ is then defined as

$$G(A) = [\log P(A)_U - \log P(A)_{agent}]^2, \tag{2.22}$$

where $P(A)_{agent}$ is the likelihood that the agent generates a SMILES through the sequence of actions A. Hence, the goal is for the agent to learn a policy that minimizes this loss, which is done through Adam.

3

Methods

With the aim of comparing the performance of the decoder and Mamba architecture with that of an RNN, the three different architectures were implemented using Python. These were all trained on two different datasets separately, one extracted from ChEMBL and one extracted from PubChem, with the aim of answering how different training datasets affect the performance of the models. The trained models were then optimized for a set of specific tasks using reinforcement learning, to study to what extent they are able to optimize for new tasks.

This chapter systematically details the methodology, starting with the preparation of training data, including its sources, pre-processing techniques, and dataset construction. It then presents the implementation of model architectures. Pretraining and reinforcement learning procedures are outlined, including scoring functions and configuration. The final section addresses specifically how the models will be evaluated to address the research questions.

3.1 The Training Data

To train the various models, two different sources of data were used and compared, namely: ChEMBL and PubChem. The raw chemical data from both of these sources consists of chemical structure data format (SDF) files from which SMILES can be extracted. The extracted SMILES are then processed through a pre-processing pipeline, removing SMILES that do not fulfill a set of criteria. The SMILES that remain after filtering are then split into training and validation sets. The entire data flow, from source to model input, is visualised in Figure 3.1. In the following sections, we aim to describe the different sources used, as well as the filtering pipeline and the preparation of data for training.

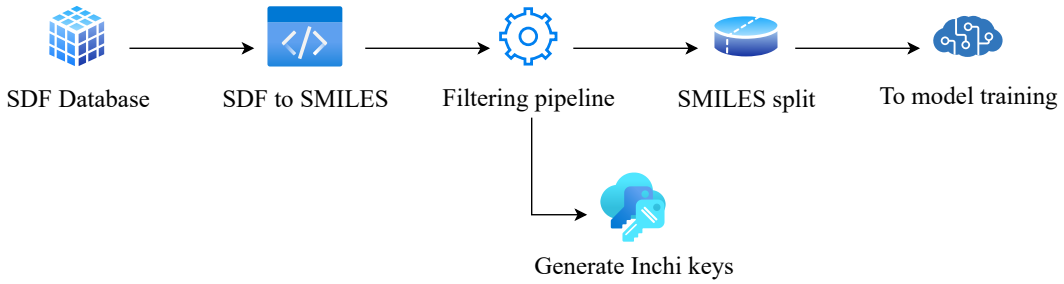


Figure 3.1: Flow of data from source to model. SMILES data is downloaded from respective database in SDF format, converted into text format or SMILES. The SMILES are filtered and then split into training and validation sets. For each of these, InChI keys are generated, and the SMILES are prepared for input to the models.

3.1.1 Data Sources

To investigate how different training data affects models' performance, SMILES data from two different sources were used to train the models: ChEMBL and PubChem. Both of these are public molecular databases containing a large number of bioactive molecules with drug-like properties. ChEMBL is developed and maintained by EMBL-EBI [16], one of six sites of the intergovernmental research organization European Molecular Biology Laboratory (EMBL). For these experiments, ChEMBL 35 was used, retrieved 2025-05-09.

PubChem is an open database from the National Institutes of Health [17], meaning that users can upload data for others to use. Pubchem was retrieved 2025-05-08.

The two datasets were downloaded in Structural Data File (SDF) format and then converted into unique SMILES strings. Duplicates were removed based on InChI keys. This conversion led to the two data sets consisting of 2.474.483 SMILES for ChEMBL and 118.638.746 SMILES for PubChem.

3.1.2 Pre-processing

The datasets were filtered according to the criteria listed in Table 3.1. This is done using a pre-processing pipeline in the Reinvent framework, which makes use of RDKit (version 2023.9.5) for filtering. In addition to the filtering, compounds are normalized to a standardized form. The standardization is aided by transformation, which maps certain substructures to standardized structures. These transformations are listed in Appendix A. The filtering also includes a control function to remove duplicate SMILES.

Table 3.1: Criteria for filtering out SMILES.

Attribute	**Threshold**	**Description**
Allowed elements	S, F, O, Cl, C, I, Br, N, B, and P.	SMILES that contain elements other than these are filtered out. These elements make up the organic subset in the original definition of SMILES and are supported without brackets in SMILES [19]. Other elements can be harmful for the human body, such as mercury [43].
Number of heavy atoms	2 to 90	Number of allowed heavy atoms, where a heavy atom is any atom that is not Hydrogen (H). The most common number of heavy atoms found in drugs are 23.5 [44]. Here, a large margin was used in order not to limit the chemical space.
Maximum molecular weight	1200 Daltons	All SMILES with a molecular weight above this threshold are excluded. Molecules weighing less than 900 Daltons are considered small, and small molecules make up 90% of pharmaceutical drugs [45]. Here, some margin is accounted for to not limit the chemical space.
Min. number of carbon atoms	2	All SMILES containing fewer than two carbon atoms are excluded; as carbon forms the backbone of organic chemistry.
Max. number of rings	12	All SMILES with more than this number of rings are removed. The vast majority of drugs contain rings [46]. Here, filtering with a high margin ensures that the chemical space is not limited.
Maximum ring size	7	Compounds containing rings with more than this number of atoms are filtered out. Sets of rings commonly seen in molecules have rings with less than 7 atoms [46].

Furthermore, the SMILES from the datasets are also transformed or retained by (1) retaining stereochemical information, (2) retaining isotope information, (3) neutralizing if possible, i.e., removing formal charges such as the charge in Na+, (4) not kekulized, meaning aromatic rings remain in their generic notation and are not forced into explicit single/double bond patterns (for example, benzene is denoted

as c1ccccc1 rather than its kekulized form C1=CC=CC=C1), and (5) not randomized, so only the canonical form of each molecule is present in the dataset. By not kekulizing SMILES, aromatic rings remain in their generic notation, and by not randomizing, only one, canonical form per molecule is present in the dataset. After filtering, the SMILES are split into training and validation sets on a 90/10 split, and InChI keys are generated for the respective sets, enabling comparison between the generated and training SMILES. InChI keys are used for comparing generated molecules, since different SMILES can describe the same molecule as described in 2.1.

For some of the experiments, the SMILES of the datasets are randomized before training. This is done for each SMILES in the training and validation set, by randomly shuffling the order in which atoms and bonds are encoded, while still representing the same chemical structure.

Before being passed to the models for training, the training and validation sets are tokenized using a vocabulary. The vocabularies are generated together with the models, and are specific to the training sets. BOS, EOS, and padding tokens are added to the vocabulary (1, 2, and 0, respectively), and the vocabularies are extended with unused tokens to give them both a final vocabulary size of 128.

During training, the data is fed to the model in batches. This requires all sequences within batches to be of the same length. Therefore, sequences are padded with padding tokens to make them equally long as the longest sequence in the batch. In addition to this, a binary padding mask is generated for each batch to tell what tokens are padding tokens. This mask is passed to the model together with each batch of sequences.

3.2 Implementation of Architectures

In the following subsections, the implemented architectures are described in detail. All architectures are implemented as a part of the Reinvent4 [47] framework, utilizing internal methods for tokenization and vocabulary creation. The decoder transformer architecture is implemented from scratch using PyTorch [48], while the RNN and Mamba architectures utilizes existing architectures.

For the baseline comparison between the RNN, the decoder, and Mamba, the architectures were implemented with the dimensions described in Tables 3.2, 3.3, and 3.4 in each architecure's subsection. By choosing these dimensions, the models have a comparable number of trainable parameters (RNN: 5.9M, decoder: 6.4M, and Mamba: 6.2M), which aligns with those used in similar experiments [49, 40].

3.2.1 LSTM RNN

The recurrent neural network was implemented using the existing RNN architecture in the Reinvent framework. This implementation builds upon the PyTorch RNNBase, which is a base class for RNN modules. For all experiments, the RNN architecture was implemented with the LSTM cell type, a layer size of 512, and a total

of 3 hidden LSTM layers. In Figure 3.2, a visualization of the implemented architecture can be seen, along with the dimensions of each layer. All model parameters are also listed in Table 3.2.

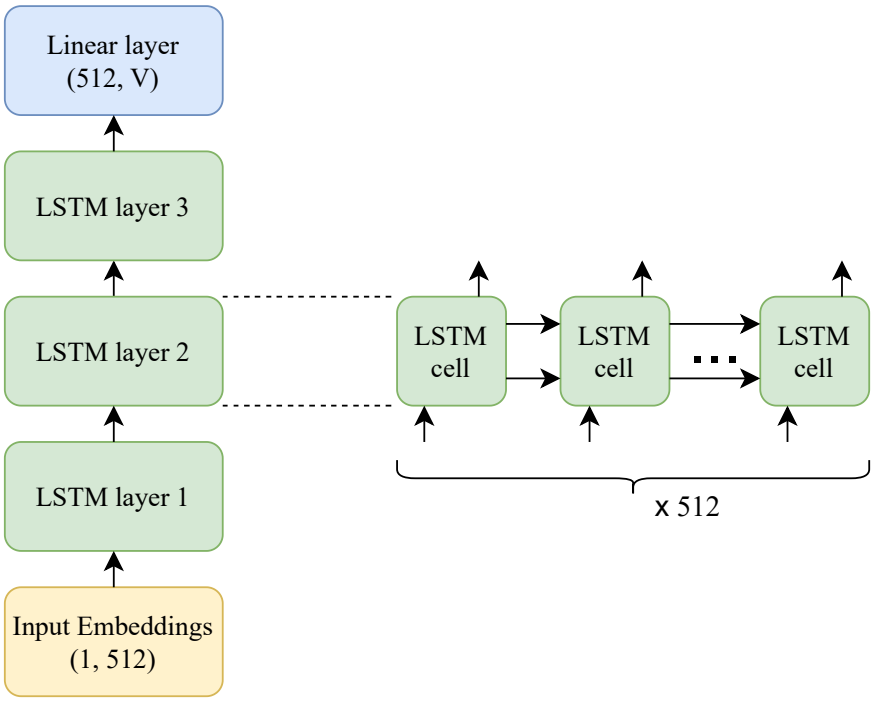


Figure 3.2: Schematic illustration of the LSTM RNN used for the experiments. Sequences of tokens are fed to the input embedding layer, which maps each token to the hidden dimension of the model, 512. The embeddings are then passed through the three LSTM layers, each with a dimension of 512 (i.e., containing 512 LSTM cells). The output of the final LSTM layer is mapped back to the size of the vocabulary, V, producing logits (unnormalized scores) for each token in the vocabulary.

RNN

Parameter	**Value**
Number of layers	3
Layer size	512
Dropout	0.0
Vocabulary size	128
Number of trainable parameters	5.9 M

Table 3.2: Architecture parameters of the RNN. The architectural layout and the parameter values amount to a total number of trainable parameters of 5.9M.

3.2.2 Decoder

The decoder was implemented using PyTorch version 2.2.1, and a visualization of the entire architecture can be seen in Figure 3.3. The architecture consists of an initial embedding layer, which maps the tokens to the dimension of the architecture, the model dimension d. This model dimension is maintained throughout the entire architecture, up until the final output layer, in which sequences are mapped back to the size of the vocabulary. The initial embedding layer is followed by a stack of decoder layers. The decoder layers employ pre-layer normalization (pre-LN), resulting in the order of action within each decoder layer seen in Figure 3.3.

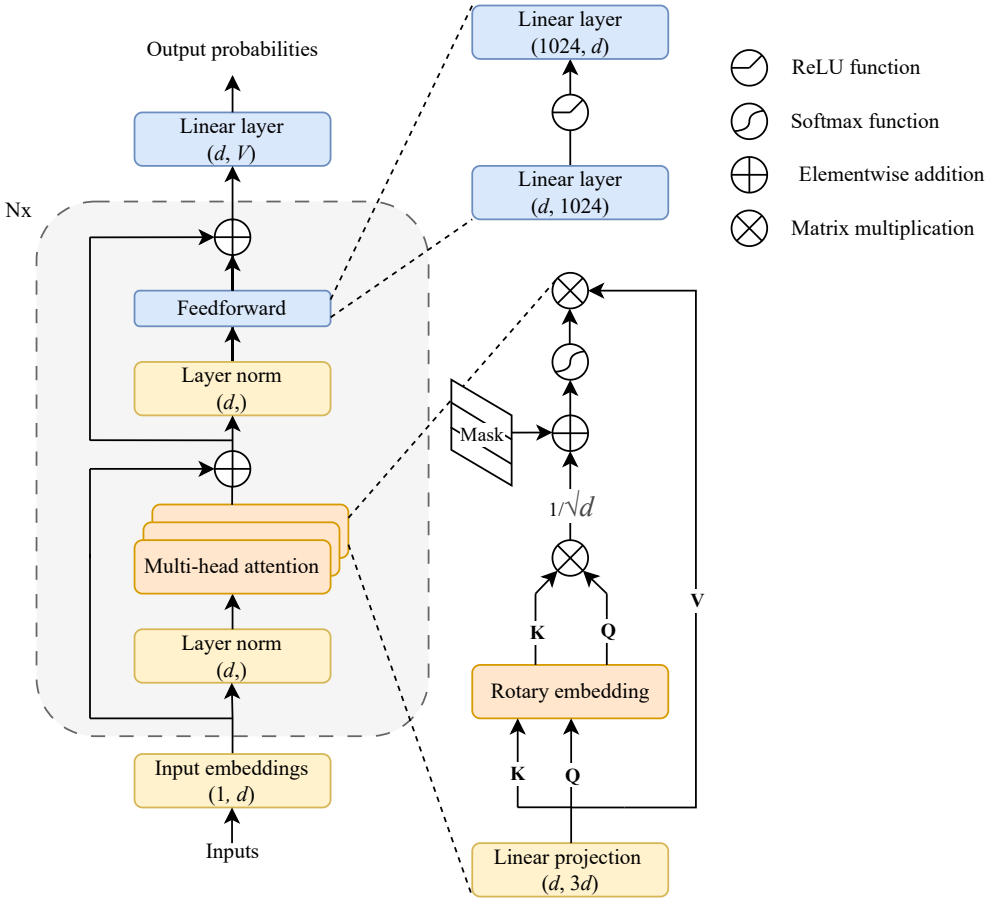


Figure 3.3: Schematic illustration of the implemented decoder architecture, detailing the workings of the multi-head attention and the feed-forward layers. Sequences of tokens are fed through an input embedding layer, which maps every token to the model dimension, d. The sequences are then processed through N decoder layers, marked by the gray area. Within each decoder layer, the sequences go through (1) layer normalization, (2) multi-head attention, (3) a second layer normalization layer, and (4) a feed-forward layer. In the detailed visualization of the multi-head attention (lower right), **K**, **Q**, and **V** refer to the Key, Query, and Value matrices.

Positional encodings are implemented as rotary positional embeddings, using the rotary-embedding-torch library with version 0.8.6 [50]. These are applied within each multi-head attention layer, rotating query and key matrices right before the scaled dot product attention. During training, masking is used within each scaled dot product attention computation. The applied mask is a combination of two different masks that serve two distinct purposes:

- **Padding mask:** This mask is passed to the model together with the batched training data and tells what tokens are padding tokens. Hence, this mask has the same shape as the input batch, i.e. (batch_size, longest_sequence_length) where longest_sequence_length refers to the length of the longest sequence of the batch. As the model should not learn to generate padding tokens, the mask is used to prevent the model from attending to these padding tokens. This mask is also used to tell what tokens should be used to compute the loss.
- **Causal mask:** This mask is generated once for each batch and is used to prevent tokens from attending to those succeeding them within a sequence. For each sequence, the causal mask is an upper triangular matrix, with shape (sequence_length, sequence_length) where the elements of the lower triangle are set to zero, and the others are minus infinity. The causal mask for the entire batch consists of stacked such triangular matrices, padded with minus infinity to the right and at the bottom, giving the entire causal mask a shape of (batch_size, longest_sequence_length, longest_sequence_length).

These two masks are combined to form an attention mask, which is added to the scaled dot product before being processed by the softmax function. The elements of the attention mask are set to 0 where the attention should be applied, and minus infinity at the positions that should be ignored by the attention mechanism.

Each multi-head attention sublayer is followed by a residual connection and normalization layer. The output of the normalization layer is fed through a dense feed-forward layer, consisting of an input layer that maps the input to the feed-forward dimension as seen in the upper right part of Figure 3.3. The ReLU activation is applied, and the input is mapped back to the model dimension in a second linear layer. The decoder layer is finalized by a second residual connection, summing the output from the feed-forward layer with the unnormalized output of the previous residual connection.

The last decoder layer is followed by a final output layer. The output layer is a simple linear layer that maps the output of the decoder layer from the model dimension back to the vocabulary dimension, producing logits (unnormalized scores) for each token in the vocabulary.

During training, dropout is applied to the input embeddings and inside the scaled dot-product attention. This means that a set of randomly sampled elements is set to zero. The elements to be zeroed are sampled independently from a Bernoulli distribution for each forward call. The amount of elements being zeroed out is defined implicitly by a probability p for each element to be zeroed. This probability p is set to 0.1 throughout all of the experiments.

The parameters of the baseline implementation of the decoder are listed in Table 3.3.

Decoder

Parameter	**Value**
Number of layers	8
Number of heads	8
Model dimension	256
Feed-forward dimension	1024
Dropout	0.1
Max. sequence length	256
Vocabulary size	128
Number of trainable parameters	6.4 M

Table 3.3: Architecture parameters of the baseline decoder. The dimension of each head is defined by the model dimension divided by the number of heads, resulting in a head dimension of 32. The architectural layout and the parameter values amount to a total number of trainable parameters of 6.4M.

3.2.3 Mamba

The Mamba (SSM) architecture was implemented using the Gu and Dao [23] Mamba framework [51] (v2.2.4) under PyTorch 2.4.1 and CUDA 12.1.1. The Mamba mixer model code provides the essential backend for language modeling, i.e., the deep sequence model backbone (with repeating Mamba blocks) and a language model head. The resulting architecture is visualized in Figure 3.4. In detail, the Mamba model consists of an initial embedding layer that maps input tokens to the model dimension d_m, followed by a stack of N Mamba layers. Each layer begins with a root mean square normalization (RMSNorm) to stabilize activations. The core of the layer is the Mamba block seen in Figure 2.11, which projects the sequence to a higher dimension d_i and applies a dynamic state space recurrence, updating states according to Equations 2.13 and 2.14. The state transition matrix $\mathbf{A}$, are initialized using a HiPPO scheme to ensure efficient memory for long-range dependencies (described by Equation 2.17). After each Mamba block a residual connection adds the layer's input to its output. The final Mamba layer is followed by RMSNorm and a linear output (language) head that maps the signal from dimension d_m back to the vocabulary size, producing per-token logits for generation.

The parameters used for the implementation of Mamba are listed in Table 3.4.

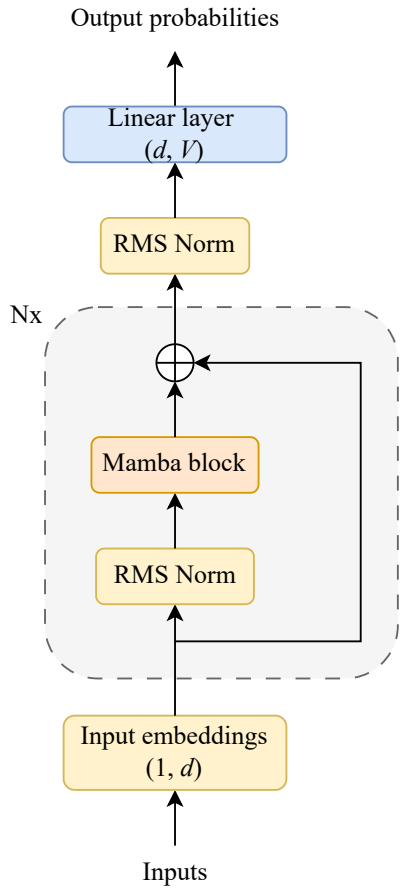


Figure 3.4: Schematic illustration of the implemented Mamba architecture, Sequences of tokens are fed through an input embedding layer, which maps each token to the model dimension, d. The sequences are then processed through N Mamba layers, marked by the gray area. Within each layer, the sequences go through RMSNorm before being passed to the Mamba block seen in Figure 2.11 and explained in Section 2.3.3.2. After the Mamba block, a residual connection from before the first normalization is added before being passed forward to another RMSNorm layer and finally through a linear layer to produce the output probabilities over the vocabulary.

Mamba

Parameter	Value
Number of layers	5
Model dimension	256
Intermediate dimension	1024
Kernel size	4
Stride	1
Padding	3
Number of trainable parameters	6.2 M

Table 3.4: Architecture parameters of Mamba. The architectural layout and the parameter values amount to a total number of trainable parameters of 6.2M. The kernel size, stride, and padding are parameters of the convolutional layer inside the Mamba block.

3.3 Pretraining Setup

The datasets are randomly split into training and validation set (90/10) for all models. During training, the different models are fed batches of tokenized SMILES string sequences, and for each of those sequences, the negative log-likelihood that the model would output that sequence is computed as per Equation 2.19. The per-batch mean of these likelihoods is then used to update the models' parameters using the Adam optimizer. The validation set is fed through the network in each epoch, and the negative log likelihood is computed. This is not used to update the models' parameters, but to make sure that the model does not overfit to the training data. At each epoch, the state of the model is saved; hence, we do not implement early stopping as part of the training, but use the validation loss to choose the best-performing parameters for our models afterwards.

To compare all architecture-dataset combinations equally, we use the same fixed training environment for all our models. The configuration for this environment has been listed in Table 3.5. Other parameters are set to PyTorch default. Keeping a fixed batch size makes the running time for an epoch scale linearly with dataset size. Therefore, the models that were trained on the larger dataset (PubChem) ran for fewer epochs than those that were trained on ChEMBL. The decoder and RNN trained on PubChem were initially trained for 15 epochs, to which an extra 10 epochs were added amounting to a total of 25 epochs. The Mamba trained on PubChem was trained for 15 epochs. The decoder and Mamba trained on ChEMBL were trained for 70 epochs, while the RNN trained on ChEMBL was trained for 50 epochs.

Parameter	Value
Batch size	50
Optimizer	Adam (PyTorch)
Initial learning rate	1×10^{-4}
LR scheduler	StepLR (`step_size` = 10, `gamma` = 0.95)
Loss	Negative log-likelihood

Table 3.5: Training parameters. All models are trained using the Adam optimizer, meaning that the learning rate adapts for each parameter. This is combined with a learning rate (LR) scheduler which decreases the learning rate by multiplication of `gamma` every `step_size` step.

The experiments have been limited so as to not include any hyper-parameter tuning, partly because the primary goals of this thesis are to evaluate inherent strengths and weaknesses of the architectures, rather than finding their optimal performance. In addition to this, hyperparameter tuning is both resource-intensive and time-consuming.

All training is done on AstraZeneca's in-house cluster using PyTorch 2.2.1 (PyTorch 2.4.1 for Mamba). For the times reported in Section 4.2.1, specifically, a NVIDIA GeForce RTX 3080 Ti with Pytorch 2.2.1 (PyTorch 2.4.1 for Mamba) was used.

3.4 Reinforcement Learning

For the task-specific optimization through reinforcement learning, the pre-trained models are used as priors and initial agent networks in the reinforcement learning setting, as visualized by Figure 3.5. A batch of SMILES is generated using the agent network. These generated SMILES are then scored using some scoring function $S(A)$ for each SMILES A, and the log likelihoods of the generated SMILES are computed for both the agent and the prior as per Equation 2.19. These are then used to compute the squared loss $G(A)$, described by Equation 2.22. The agent network is updated through gradient descent (e.g., Adam optimizer) to minimize the loss $G(A)$. This process is repeated through a fixed number of steps using different scoring functions.

3.4.1 Scoring Functions

In the reinforcement learning setting, the scoring function $S(A)$ determines which molecular properties are desirable. In this project, the experiments will be run for five different optimization tasks, one at a time, each time starting from a prior trained on a large set of SMILES (from ChEMBL or PubChem). The five optimization tasks and corresponding scoring functions used are described below.

Sulphur avoidance. By penalizing SMILES containing sulphur, denoted by S, the goal is to generate sulphur-free molecules. This task is fairly simple, and rather

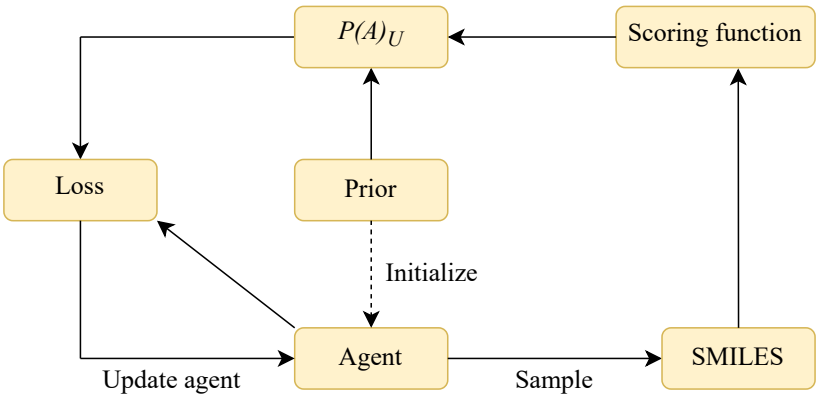


Figure 3.5: The reinforcement learning loop. An agent network is initialized from the prior, and this agent is then used to sample a batch of SMILES. The SMILES are scored using the scoring function, and the scores are combined with the NLL from the prior for the sampled SMILES, forming the augmented likelihood $P(A)_U$ as per Equation 2.5 of the sampled SMILES. The loss is computed as the square loss between the augmented likelihood and the NLL from the agent, and is used to update the agent network through stochastic gradient descent.

than being an actual task that the models would be used for, it acts as a proof of principle. The scoring function is simply

$$S(A) = \begin{cases} 1 & \text{if valid SMILES and no S} \\ 0 & \text{if not valid SMILES} \quad \textit{or} \quad \text{if contains S.} \end{cases} \tag{3.1}$$

Similarity guided structure generation. For this task, the purpose is to optimize the agent to generate a structure similar to a query structure. Based on previous work by Olivecrona et al. [40], the structure chosen as query structure is Celecoxib, which is a drug that targets the enzyme COX-2, responsible for pain and inflammation in the human body [52]. To measure the similarity between the Celecoxib molecule and the generated structures, the Jaccard index with Morgan fingerprints is used. The Jaccard index is defined as the intersection of two samples A and B over their union, i.e,

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}, \tag{3.2}$$

and is often referred to as the Tanimoto similarity or Tanimoto index. Rather than comparing single atoms between two structures, it's common to use chemical fingerprints that represent substructures of a molecule [53]. The size of the substructures is governed by the radius of the fingerprint, which is set to a value of 2 in line with what was done by Olivecrona et al. [40]. Denoting the Jaccard index between two compounds i and j as $J_{i,j}$ the scoring function takes the form of

$$S(A) = J_{i,j}. \tag{3.3}$$

Target activity: Dopamine Receptor D2 (DRD2). To optimize generated structures to be active against the dopamine type 2 receptor (DRD2), a key target for antipsychotic drugs [54], a support vector machine classifier is used to predict a generated compound's DRD2 activity [40]. This predictive model outputs an unnormalized probability of being active, p_{active}. Hence, the scoring function takes the form of

$$S(A) = p_{active}. \tag{3.4}$$

Target activity: c-Jun N-terminal Kinases-3 (JNK3). This mitogen-activated protein kinase is responsive to stress stimuli and is found mainly in the brain. Similar to the case for DRD2, a classification model developed by Li, Zhang, and Liu [55] is used to predict the generated compounds' activity towards JNK3. This classifier is a random forest classifier that, just like the DRD2 classifier, outputs an unnormalized probability of being active, p_{active}, which is used as the scoring function.

Target activity: Glycogen Synthase Kinase 3 Beta (GSK3β) This is a serine/threonine protein kinase which has been identified as a protein kinase for over 100 different proteins and is hence implicated in a number of various diseases such as diabetes, cancer, and bipolar disorder. The target activity of a compound towards GSK3β is also predicted using a random forest classifier developed by Li, Zhang, and Liu [55] and the output of the classifier is used in the same manner as the other target activity scoring functions.

3.4.2 Reinforcement Learning Setup

The pretrained priors are trained through reinforcement learning for each of the scoring functions listed above. Starting from a pretrained model, the reinforcement learning is run for 1000 steps for each model and scoring function, at each step sampling a batch of 64 molecules. All molecules throughout all steps is collected in a technique called enhanced sampling, where the generated molecules during reinforcement learning are all relevant. To promote diversity among generated compounds, a diversity filter (DF) based on scaffold diversity is utilized. The filter has a memory that is organized into buckets, where each bucket holds a specific scaffold. Each bucket has a limited size, and when a bucket is filled, every further occurrence of that scaffold will be given a zero score. The Reinvent framework supports different types of scaffolds, and for this experiment, the Murcko scaffold is used, which removes all side chains from a molecule. This diversity filter is applied globally, meaning that it operates over the entire reinforcement learning run and all compounds generated under it. The parameter settings of this diversity filter are listed in Table 3.6 along with other parameter settings for the reinforcement learning. The diversity filter is used for all optimization tasks except for similarity guided structure generation, for which recurring SMILES are penalized by a factor of 0.5 instead.

Parameter	Value
Batch size	64
Number of steps	1000
Optimizer	Adam
Initial learning rate	0.0001
σ (in Equation 2.21)	128
(DF) Scaffold type	Murcko Scaffold
(DF) Bucket size	25

Table 3.6: Reinforcement Learning Parameters. The σ parameter refers to the constant used to weight the score $S(A)$ in the computation of the augmented log likelihood in Equation 2.21. The parameters prefaced with "(DF)" refer to parameters of the diversity filter.

The task-specific scoring function is combined with Quantitative Estimation of Drug-likeness (QED) [56], forming the total score as the geometric mean (with unit weights) over the task-specific score and the QED score. It is this total score, $S(A)$, that is being used to compute the loss and update the network parameters. The QED is an aggregation of a set of different molecular descriptors, such as molecular weight and topological surface area, and it aims to reflect the underlying distribution of molecular properties.

The results vary between reinforcement learning runs as it is heavily dependent on the compounds generated by the model at each step; for this reason, the reinforcement learning was run three times for each prior and each optimization task. With several runs, the results could be averaged, and the standard deviation over runs estimated.

3.5 Evaluation

The implemented architectures were evaluated on a set of metrics. For the pre-training, where the purpose of the training is for the models to produce drug-like compounds representative of the training data, a set of 10^4 SMILES was sampled from each model at every epoch, and for each generated SMILES it was checked to be (1) *valid* in that way that RDKit is able to create a molecule object from it, (2) *unique* in that it has not been generated by the model before, and (3) *novel*, meaning that it does not exist in the training data. For the latter of these two, uniqueness and novelty, InChI keys were used to compare generated SMILES in a standardized form.

To further investigate the architectures, the sampled molecules were studied with respect to some common descriptors: QED, number of heavy atoms, and the number of rings per molecule.

The Quantitative Estimate of Drug-likeness (QED) [56] is a continuous metric designed to assess how drug-like a molecule is based on the desirability of its physico-

chemical properties as observed in approved oral drugs. QED aggregates key molecular descriptors such as molecular weight, or number or aromatic rings d_i. The QED score for a molecule is computed as the geometric mean of these desirability functions:

$$\mathrm{QED} = \left(\prod_{i=1}^{n} d_i \right)^{1/n}, \tag{3.5}$$

or equivalently as an exponential of the mean logarithm

$$\mathrm{QED} = \exp\left(\frac{1}{n} \sum_{i=1}^{n} \log d_i \right). \tag{3.6}$$

Values range from 0 (least drug-like) to 1 (most drug-like), allowing for ranking of molecules by their overall suitability as drug candidates.

Comparing these distributions to those of the training data provides insight into the models' ability to "learn the data" on which it has been trained. To quantitatively compare the distributions of molecular descriptors (QED, molecular weight, ring count) between the generated molecules and the training data, the Kullback-Leibler (KL) divergence was computed. A lower KL divergence indicates that the generative model more closely reproduces the statistical profile of the training set, whereas a higher KL divergence signals greater deviation.

For reinforcement learning, the task-specific scoring function provides a direct measure of the model's ability to learn the new task. In addition to this, the QED was used as an additional scoring function, providing insight into the drug-like properties of generated compounds. However, it is also of importance that the model continues to generate molecules with a high variety between compounds, hence the variation was studied through a set of different metrics: for compounds generated during RL, we computed the number of unique comounds, the number of unique Murcko scaffolds, as well as the number of unique genericized Murcko scaffolds. While reinforcement learning is commonly used to fine-tune a model, in this case, it is applied primarily for enhanced sampling, generating a wide array of molecular candidates as the model explores chemical space. Because of this approach, we focus on analyzing the diversity and properties of compounds produced *throughout* the RL training process, rather than solely evaluating a fine-tuned models final output.

The purpose of studying these specific metrics is to provide a nuanced answer as to how a decoder and Mamba architecture compares to the RNN in terms of learning the chemical space and to what extent it can adapt to new tasks through reinforcement learning. For the architecture to be useful, it should first of all generate good drug candidates, i.e., good in the sense that it fulfills the criteria defined through scoring functions. However, it should also generate a high variety of new and unseen compounds. By studying the metrics listed above for models trained on different datasets, we also get a direct comparison of how the different datasets affect the architectures.

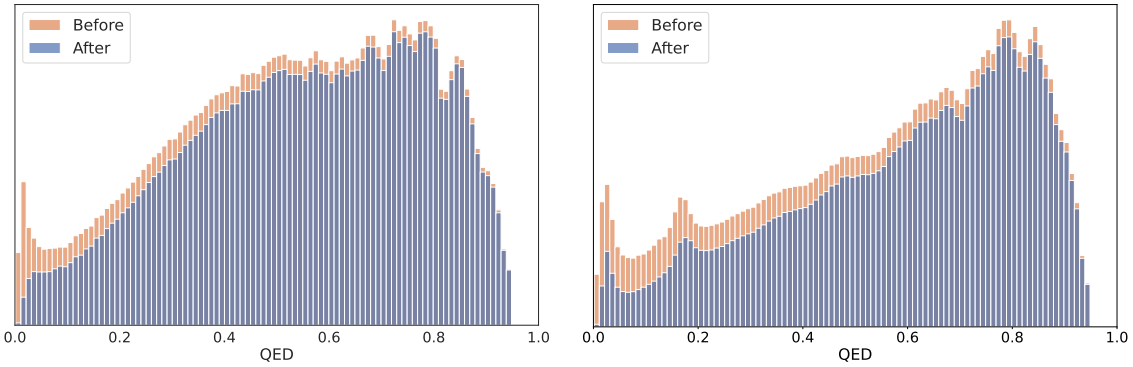
4

Results

4.1 Statistics of the Training Data

When filtering the datasets through the preprocessing pipeline described in Section 3.1.2, the ChEMBL dataset was reduced from 2.474.483 SMILES to 2.265.386 SMILES, corresponding to a reduction by 8.5 percent. The PubChem dataset was reduced from 118.638.746 SMILES to 101.869.507 SMILES, corresponding to a reduction by 14 percent. The datasets were then split into training and validation sets through a 90/10 split. The training and validation sets were created by random sampling of the datasets using the train_test_split method in scikit-learn version 1.2.2. For ChEMBL, the training data set amounted to 2.038.847 SMILES and the validation set to 226.539 SMILES. For PubChem, this amounted to 91.682.556 SMILES and 10.186.951 SMILES for the training and validation sets, respectively.

After pre-processing, the filtered SMILES were analyzed for Quantitative Estimation of Drug-likeliness (QED), which is a common measurement in drug design based on the molecular properties [56] (see details in Section 3.5). A comparison of the distribution of QED for the data before and after pre-processing can be seen in Figure 4.1. For both datasets, the distribution of the QED before is similar to the distribution after preprocessing. For the ChEMBL dataset, there is a peak of compounds with a QED close to 0, which is removed through the preprocessing. For both the ChEMBL and PubChem datasets, more compounds with a low QED are removed, and fewer with a larger QED. The ChEMBL dataset contains a larger fraction of compounds with a QED > 0.5 (both before and after preprocessing) than the PuBChem dataset, which aligns with ChEMBL's bioactive focus for drug-like molecules and PubChem's broader range of chemical structures.

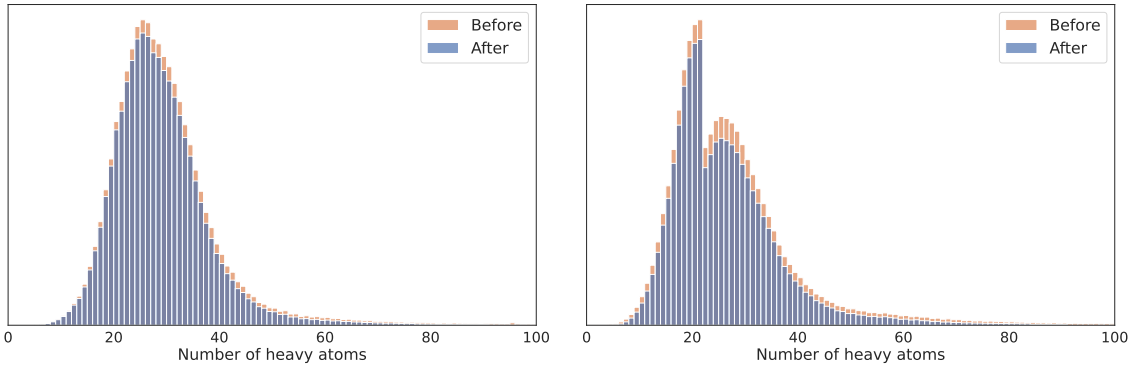


(a) Distribution of QED for ChEMBL.

(b) Distribution of QED for PubChem.

Figure 4.1: Distribution of QED before (orange) and after (blue) preprocessing. Both datasets contain a larger fraction of compounds with a QED > 0.6, and a large fraction of the removed compounds have a QED < 0.7. ChEMBL contains a larger fraction of compounds with a QED > 0.5 compared to PubChem.

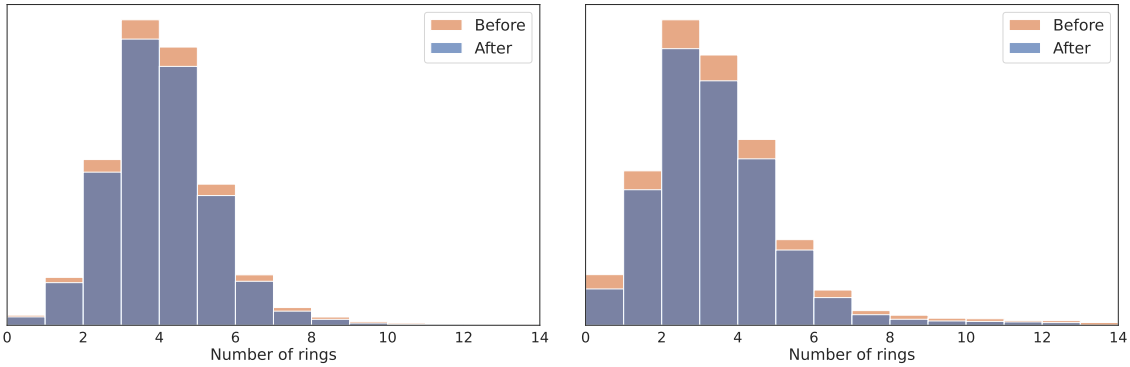
In Figure 4.2, the distribution of the number of heavy atoms in both datasets before and after preprocessing can be seen. In contrast to the QED, which the datasets are not filtered by, the number of heavy atoms is a metric that is explicitly filtered by, as explained in Section 3.1.2. As described in Table 3.1, only SMILES with a number of heavy atoms within the range of 2 to 90 are kept after filtering. As per Figure 4.2, the amount of SMILES filtered out based on the number of heavy atoms is low. The two datasets differ in the shape of the distributions, where PubChem has a spike in compounds with ~ 20 heavy atoms, both before and after preprocessing. This spike declines abruptly after 20 to continue to follow a normal distribution with a mean $\mu \simeq 25$. For ChEMBL, the distribution follows a normal distribution with $\mu \simeq 25$.



(a) Number of heavy atoms for ChEMBL. (b) Number of heavy atoms for PubChem.

Figure 4.2: Distribution of number of heavy atoms before (orange) and after (blue) preprocessing. As per Table 3.1, the SMILES are filtered out explicitly based on the number of heavy atoms. SMILES with a number of heavy atoms outside the range [2, 90] are filtered out. The distribution shows that not many SMILES are filtered out based on this criterion. The two datasets differ in the shape of the distribution. PubChem has a spike of compounds with a number of heavy atoms ~ 20 to then follow a normal distribution with $\mu \simeq 25$, while ChEMBL follows a normal distribution with $\mu \simeq 25$.

In Figure 4.3, the distribution of the number of rings per compound before and after preprocessing can be seen. This is another metric which has been explicitly filtered according to Table 3.1. SMILES having more than 12 rings are removed through filtering, and as per Figure 4.3, the filtering based on this criterion does not impact the distributions notably. In PubChem, the compounds generally have fewer rings than in ChEMBL. In PubChem, the most common number of rings among the compounds is 2, while for ChEMBL this is 3.



(a) Number of rings per compound for ChEMBL.

(b) Number of rings per compound for PubChem.

Figure 4.3: Distribution of number of rings per compound before (orange) and after (blue) preprocessing. As per Table 3.1, the SMILES are filtered out explicitly based on the number of rings per compound. SMILES with a number of rings above 12 are filtered out. The distribution before and after shows that not many SMILES are filtered out based on this criteria. The most common number of rings in PubChem is 2 while for ChEMBL this number is 3.

Other descriptors, such as molecular weight and number of carbon atoms, were computed for the two datasets before and after preprocessing; distributions of these can be found in Figure B.1 of Appendix B.1. None of the distributions of these change dramatically as a result of the preprocessing; instead, the filtering component responsible for removing a majority of compounds is the allowed elements component.

4.2 Pretraining Results

In the following sections, the results from pretraining across the three architectures RNN, decoder-only Transformer, and Mamba, as explained in Section 3.3 are presented. The datasets were split into training and validation set (90/10) for all models. During training, the models were fed batches of tokenized SMILES string sequences, computing the negative log-likelihood the model would output for each sequence. The per-batch mean of these likelihoods were then used to update the models' parameters using the Adam optimizer. The validation set was fed to the network in each epoch, computing the negative log-likelihood to control for overfitting, i.e., the validation set was not used in training the models. The state of the models was saved at every epoch and evaluated after training. In addition to comparing pretraining on the two datasets, ChEMBL and PubChem, the effect of pretraining using randomized SMILES is studied for all three architectures. Finally, a study of the impact of parameter count on the decoder was conducted. In addition to the results presented here, a deeper analysis of the distributions of descriptors for compounds generated by the pretrained models was performed. The results of this analysis can be found in Appendix B.2.

4.2.1 Training Process

The implemented models based on the architectures described in Section 3.2 were trained on the two different datasets for a varying number of epochs using the negative log-likelihood (Equation 2.19) as loss function. The resulting negative log-likelihood for each model can be seen in Figure 4.4. While all models reach a negative log-likelihood of approximately the same size for the validation loss, it is important to note that these validation losses have been computed across two different validation sets (one from ChEMBL and one from PubChem). The decoder trained on ChEMBL reaches a lower validation loss than the RNN but similar to the lowest loss for Mamba trained and validated on the same dataset. For the RNN and Mamba trained on ChEMBL, the validation loss stops decreasing after 20 and 16 epochs, respectively, and then begins to increase, while the training loss continues to decrease. This indicates that the models are not only learning the underlying patterns in the training data, but are also fitting to dataset-specific patterns, which do not generalize to unseen data such as the validation set. This issue is known as overfitting, and continuing training under these circumstances generally leads to poorer model performance. The models trained on PubChem do not exhibit overfitting. Instead, they continue to improve over all epochs. Here, the models were trained until validation loss stopped decreasing significantly. Computational constraints prevented the models from reaching total convergence. Mamba trained on PubChem reaches a lower validation loss than both the decoder and RNN trained on the same dataset.

During training, validation loss is computed after each epoch's parameter updates, resulting in initially lower validation loss compared to training loss due to rapid loss reduction in early epochs. For the models trained on PubChem, the losses start at a loss of 27.5, whereas for the ChEMBL models, it takes 2 to 3 epochs to reach this loss. This is due to that during one epoch, the PubChem models see more data, and the network is updated more times (since the batch size is fixed). Overall, the decrease in loss over the epochs is more rapid for the models trained on ChEMBL than for those trained on PubChem.

The RNN and Mamba trained on ChEMBL achieve their minimum validation loss at epoch 20 and 16, respectively, whereas all other models reach their lowest validation loss at the final epoch. For subsequent experiments, model weights were selected from the epoch with the minimum validation loss for each model.

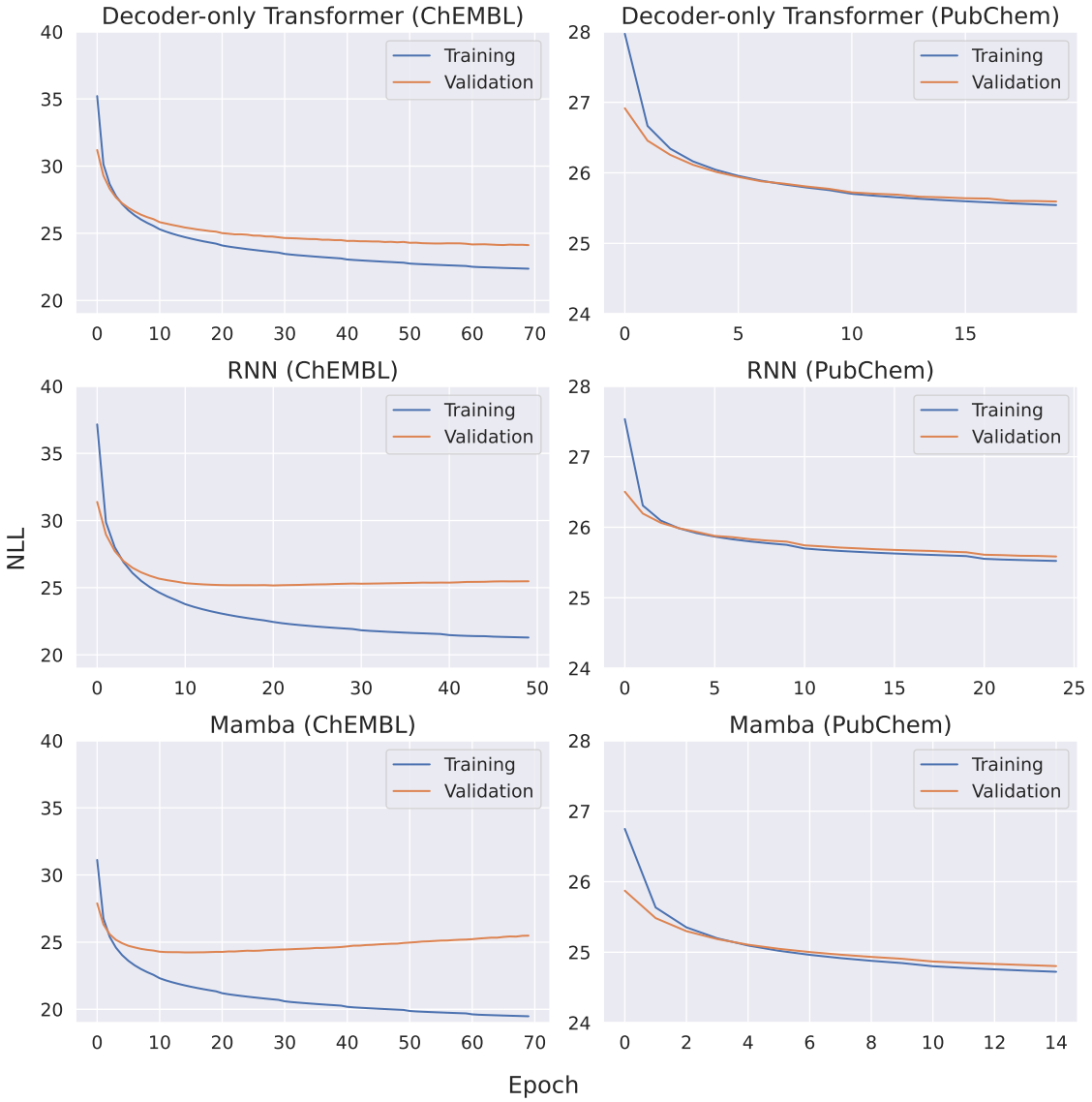


Figure 4.4: Training and Validation losses over training epochs for the decoder, RNN, and Mamba trained on ChEMBL and PubChem datasets. Note that each model was trained for various numbers of epochs: the decoder was trained for 70 epochs when training on the ChEMBL dataset, and 20 epochs when being trained on the PubChem dataset. The RNN was trained for 50 epochs when training on the ChEMBL dataset, and 25 epochs when being trained on the PubChem dataset. The Mamba was trained for 70 epochs when training on the ChEMBL dataset, and 15 epochs when training on the PubChem dataset.

To compare the computational times across architectures and datasets, all models were trained once on a NVIDIA GeForce RTX 3080 Ti GPU. The times recorded on average per epoch are listed in Table 4.1. Note that these times are the time it takes for the model to process the entire training and validation set once, with a batch size of 50, for both training and validation sets. When comparing the training times per epoch on the two separate datasets, the decoder-only Transformer requires twice

as long as the RNN, while the Mamba architecture takes approximately 1.5 times longer than the RNN.

	Time per epoch
Decoder - ChEMBL	1414 s (23 min 34 s)
RNN - ChEMBL	698 s (11 min 38 s)
Mamba - ChEMBL	1080 s (18 min)
Decoder - PubChem	61936 s (17 h 12 min)
RNN - PubChem	31234 s (8 h 40 min)
Mamba - PubChem	48576 s (13 h 30 min)

Table 4.1: Training times for all model-dataset combinations. Note that these times are only for training the model for one epoch, processing the entire training and validation set (but not the sampling described in Section 4.2.2).

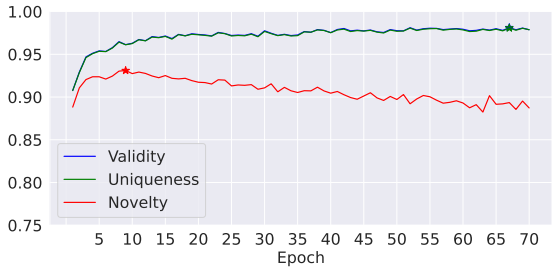
4.2.2 Validity, Uniqueness, and Novelty

For each epoch during training, the validity, uniqueness, and novelty were computed for each model by sampling a set of 10^4 molecules. For each compound, it was determined whether it was (1) valid in the sense that it was possible to create a molecule object from the SMILES string using RDKit, (2) unique among the sampled molecules so that none of the other sampled molecules that epoch had the same InChI key, and (3) novel with regards to the InChI keys of the molecules in the training data. The number of valid, unique, and novel molecules was normalized by the total number of sampled molecules to derive the validity, uniqueness, and novelty. The resulting scores for each model can be seen in Figure 4.5. Overall, the validity and uniqueness follow each other for all models, making them nearly indistinguishable in the figures, which means that a majority of the valid compounds generated do not repeat. While the validity and uniqueness are high and similar for all models trained on the PubChem dataset, the decoder trained on ChEMBL has a higher validity and uniqueness compared to the RNN and Mamba trained on the ChEMBL dataset. Mamba and the RNN exhibit similar validity and uniqueness across epochs for ChEMBL as well as for PubChem.

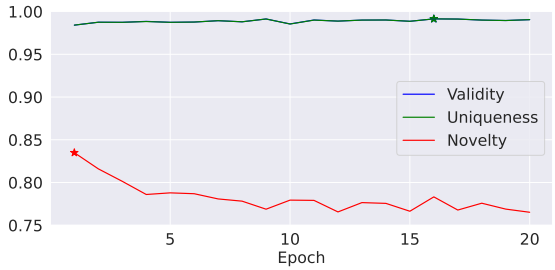
For all models, novelty declines over training steps, with maximum novelty observed within the first ten steps for all ChEMBL models, and at the first step for all PubChem models. This decline reflects the models' increasing tendency to reproduce examples from the training set as learning progresses, resulting in fewer novel outputs. Additionally, the novelty is significantly lower for PubChem models, likely due to the much larger size of the PubChem training set used for novelty computations, which increases the probability of generating molecules already present in the training data.

As observed with the losses in Section 4.2.1, the validity, uniqueness, and novelty

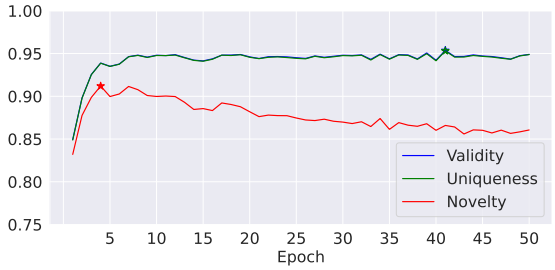
metrics remain more stable for PubChem-trained models compared to those trained on ChEMBL. PubChem-trained models exhibit a high fraction of valid and unique SMILES after the first epoch, likely because a single epoch entails more weight updates than for the ChEMBL-trained models.



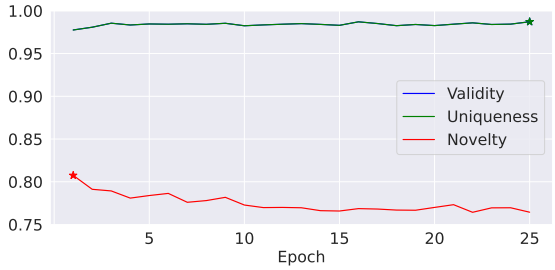
(a) Decoder (ChEMBL)



(b) Decoder (PubChem)

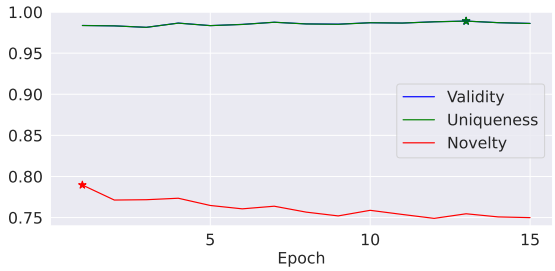


(c) RNN (ChEMBL)



(d) RNN (PubChem)

(e) Mamba (ChEMBL)



(f) Mamba (PubChem)

Figure 4.5: Validity, uniqueness, and novelty of the decoder, RNN, and Mamba trained on the ChEMBL or Pubchem dataset over epochs. The highest scores have been indicated by stars for each metric. Note that validity and uniqueness follow each other closely for all models, making them difficult to distinguish. The models exhibit similar behavior on the same dataset, but the decoder trained on ChEMBL exhibits higher validity and uniqueness than the RNN and Mamba trained on ChEMBL.

The validity, uniqueness, and novelty assessed at the epochs corresponding to the lowest validation loss are presented in Table 4.2. Notably, these values do not represent the maximum achieved for any metric during training; this is particularly evident for novelty, which peaked earlier in the training process for all models. However, for the validity and uniqueness, the listed values are representative of the best performance of the models.

Training with the PubChem dataset led to increased validity and uniqueness for all models. The decoder trained on PubChem achieved the highest validity and uniqueness of all models, with identical fractions for valid and unique molecules (0.9905). Similarly, for the PubChem-trained RNN and Mamba their corresponding validity and uniqueness are equal (0.9872 and 0.9890 respectively), i.e., all valid molecules generated by these models are also unique. When trained on the same dataset, the decoder outperforms the RNN and Mamba with respect to validity and uniqueness across both ChEMBL and PubChem, but Mamba has the highest novelty across both datasets. This could, however, be due to Mamba being trained for fewer epochs (15) relative to the Decoder (25) and the RNN (20).

	Validity	**Uniqueness**	**Novelty**
Decoder - ChEMBL	0.9787	0.9634	0.8872
RNN - ChEMBL	0.9461	0.9456	0.8819
Mamba - ChEMBL	0.9482	0.9477	**0.8899**
Decoder - Pubchem	**0.9905**	**0.9905**	0.7652
RNN - Pubchem	0.9872	0.9872	0.7644
Mamba - Pubchem	0.9890	0.9890	0.7896

Table 4.2: Validity, uniqueness, and novelty metrics for the decoder, RNN, and Mamba models trained on either the ChEMBL or PubChem dataset. These metrics were calculated by sampling a set of 10^4 molecules from each model, using the weights from the epoch with the lowest validation loss. For the RNN and Mamba models trained on ChEMBL, this corresponds to the 20th and 16th epochs, respectively; for all other models, metrics are computed using the final epoch.

4.2.3 Effect of Randomized Smiles

By randomizing the SMILES representations in the training dataset prior to model input, each canonical SMILES entry was replaced with a randomized, yet valid, alternative encoding of the same molecule. This randomized SMILES was generated by randomly traversing the molecular graph to produce a non-canonical string representation. This procedure yielded a new dataset containing the same molecules but with different SMILES strings, introducing variation in the underlying chemical structural patterns presented to the model.

Figure 4.6 presents the loss curves for models trained with randomized SMILES across epochs. Both the RNN and Mamba models exhibit overfitting to the training data after 14 and 10 epochs, respectively. As shown in Figure 4.6, the lowest loss for all models trained on the PubChem dataset occurred at the final epoch. In contrast, for models trained on the ChEMBL dataset, the epoch corresponding to the lowest loss varied as follows: Decoder at epoch 66, RNN at epoch 14, and Mamba at epoch 10.

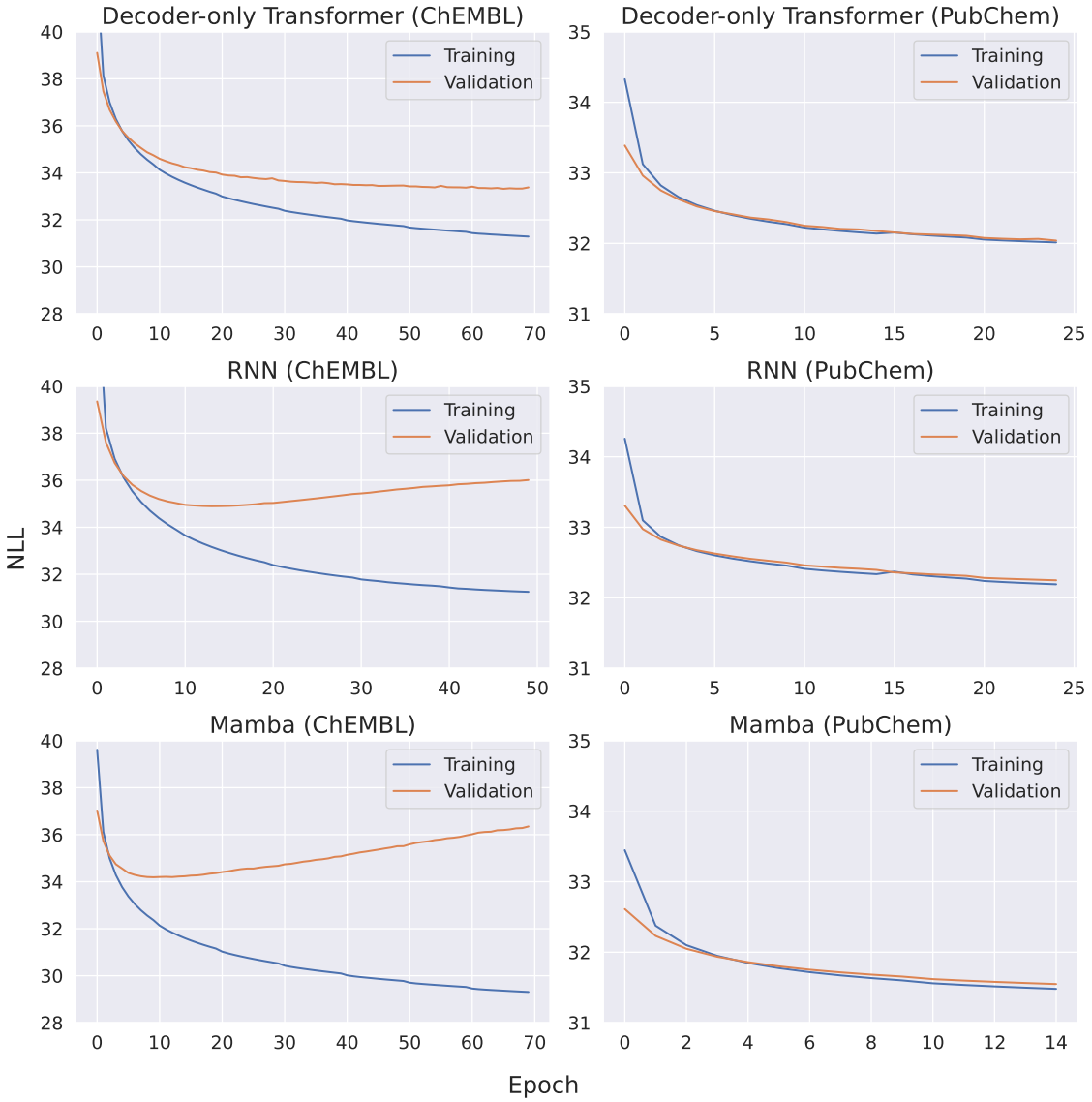
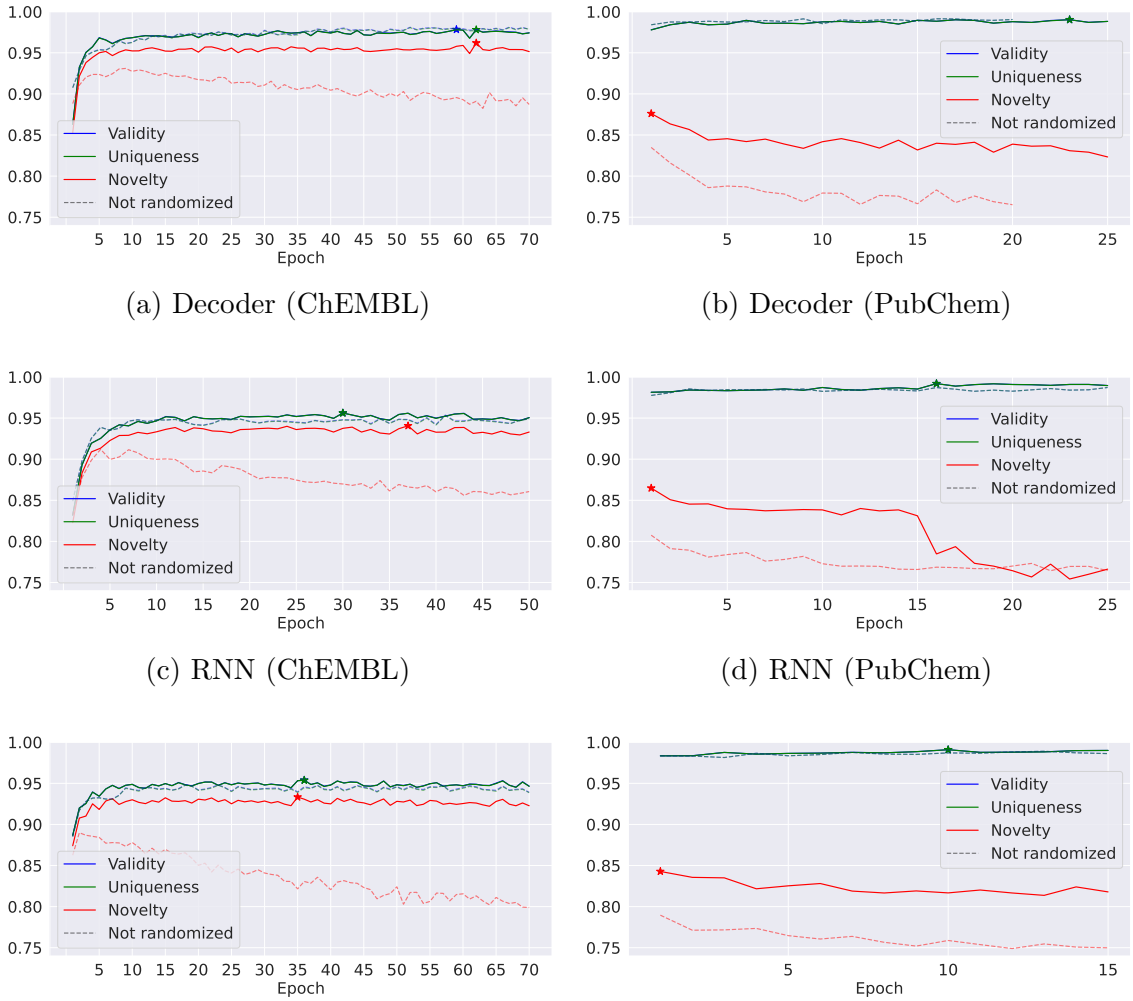


Figure 4.6: Training and validation losses over training epochs when training with randomized SMILES. Note that each model was trained for various numbers of epochs: The decoder was trained for 70 epochs when training on the ChEMBL dataset, and 25 epochs when being trained on the PubChem dataset. The RNN was trained for 50 epochs when training on the ChEMBL dataset, and 25 epochs when being trained on the PubChem dataset. The Mamba model was trained for 70 epochs when training on the ChEMBL dataset, and 15 epochs when training on the PubChem dataset.

Figure 4.7 shows the validity, uniqueness, and novelty (see Section 4.2.2 for how these are computed) over epochs for the models trained with randomized SMILES. The highest value for each metric in each subplot has been marked with a star. For validity and uniqueness, this value often coincides, leading the subplots to include a shared star for these metrics. The dashed lines represent the validity, uniqueness, and novelty for the models trained on canonical SMILES (non-randomized), and are

colored according to the metrics.



(a) Decoder (ChEMBL)

(b) Decoder (PubChem)

(c) RNN (ChEMBL)

(d) RNN (PubChem)

(e) Mamba (ChEMBL)

(f) Mamba (PubChem)

Figure 4.7: Validity, uniqueness, and novelty over epochs for models trained with randomized SMILES. The highest value for each metric has been marked with a star. For most models and datasets, the highest value for validity and uniqueness coincides, leading to them sharing the marked star. The dashed line for each metric indicates those from the models trained on canonical SMILES. All models trained with randomized smiles exhibit higher performance in novelty compared to their canonical counterparts. Note that the decoder trained on PubChem has been trained for 5 more epochs than its canonical counterpart.

All models trained on ChEMBL show similar behavior: a dramatic increase in all metrics during the first five epochs, which then converges and oscillates around the convergence limit. Similar to the models trained with canonical SMILES, the validity and uniqueness follow each other across epochs, with the decoder achieving a higher validity and uniqueness. Different from the models trained on canonical SMILES, the models trained with randomized SMILES do not exhibit a decrease in novelty but rather the opposite; the novelty continues to increase beyond the novelty

peak for canonical SMILES-trained models.

The models trained on PubChem also show similar behavior to corresponding canonical SMILES-trained models: High and almost static validity and uniqueness, and a novelty significantly lower than corresponding ChEMBL-trained models. Comparing the PubChem models trained with randomized SMILES with corresponding models trained with canonical SMILES (dashed lines in Figure 4.7), the novelty is higher for the models trained with randomized SMILES. This is true for all three PubChem-trained models across epochs, except for the RNN, which drops in novelty after 15 epochs.

In Table 4.3, the validity, uniqueness, and novelty from the epochs corresponding to the lowest validation loss (see Figure 4.6) are shown. These values do not represent the maximum achieved for any metric during training.

	Validity	**Uniqueness**	**Novelty**
Decoder - ChEMBL	0.9759	0.9750	**0.9563**
RNN - ChEMBL	0.9465	0.9464	0.9326
Mamba - ChEMBL	0.9488	0.9487	0.9300
Decoder - Pubchem	0.9882	0.9882	0.8233
RNN - Pubchem	0.9896	0.9895	0.7662
Mamba - Pubchem	**0.9900**	**0.9900**	0.8350

Table 4.3: Validity, uniqueness, and novelty for the decoder, RNN, and Mamba trained on the ChEMBL or PubChem dataset with randomized SMILES. The metrics were computed by sampling a set of 10^4 molecules from the models with the weights from the epoch with the lowest validation loss. For the decoder, RNN, and Mamba trained on ChEMBL, it corresponds to the 66th, 14th and 10th epoch, whereas for all other models it is the final epoch.

The Mamba trained on PubChem has the highest validity and uniqueness overall, while also outperforming the decoder and RNN in novelty on that dataset. For the ChEMBL dataset, the decoder outperforms the RNN and Mamba. The decoder trained on ChEMBL also has the highest novelty overall.

Comparing the validity, uniqueness, and novelty reported in Table 4.3 with that of the corresponding models trained with canonical SMILES, reported in Table 4.2, the models trained on randomized SMILES showed especially better performance with respect to novelty than their canonical counterparts. For these reasons, the models trained on randomized SMILES were included in the reinforcement learning experiments in Section 4.3.

4.2.4 Impact of Decoder Downsizing

In addition to the 6M parameter-sized decoder, a smaller version (called Decoder Mini) was designed and trained in the same manner as the baseline decoder models (with canonical SMILES). The Decoder Mini was designed with 4 decoder layers with 6 heads each, instead of the 8 layers with 8 heads of the baseline decoder. In order to keep the head dimension of 32 (as for the baseline Decoder model), the Decoder Mini was designed with a model dimension of 192. The total trainable parameter count amounted to $\sim 2.2M$. This smaller decoder was trained on ChEMBL and PubChem for 70 and 20 epochs, respectively. Training the Decoder Mini on ChEMBL for one epoch took 589 s (1414 s for baseline decoder on ChEMBL), and 26393 s (61936 s for baseline decoder on PubChem) when training on PubChem. Comparing these times with those of the other models, listed in Table 4.1, the Decoder Mini takes approximately the same time to train as the RNN.

The resulting loss curves when training the Decoder Mini can be seen in Figure 4.8. Both when training on ChEMBL and PubChem, the loss curves continue to decrease over all epochs. Comparing these losses with those seen for the baseline decoder in Figure 4.4, the Decoder Mini does not reach as low loss values as the baseline decoder.

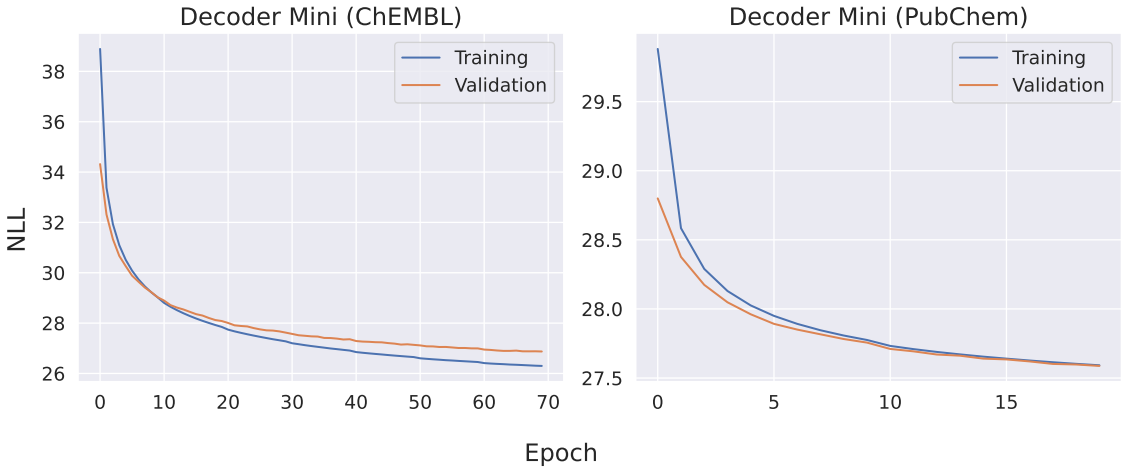
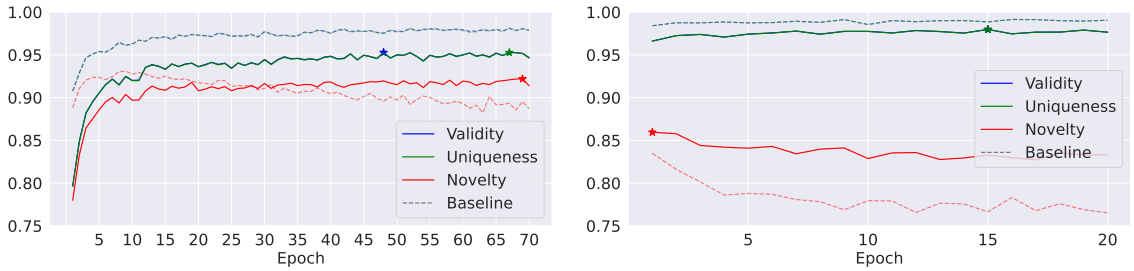


Figure 4.8: Training and validation losses over training epochs for the Decoder Mini trained on ChEMBL and PubChem datasets.

As for the other evaluated models, the validity, uniqueness, and novelty were computed for 10^4 compounds sampled at each epoch, and the results of this can be seen in Figure 4.9 together with the validity, uniqueness, and novelty of corresponding baseline decoders. The Decoder Mini trained on ChEMBL achieves a higher novelty than the one trained on PubChem; however, training the Decoder Mini on PubChem does result in a higher validity and uniqueness. For the Decoder Mini trained on ChEMBL, the validity and uniqueness demonstrate similar change over epochs as the baseline decoder trained on the same dataset (marked by dashed lines in 4.9a. The validity and uniqueness are, however, slightly lower, around 0.95, than the validity and uniqueness of the baseline decoder (around 0.97, see Table 4.2). The novelty,

which initially is lower for the Decoder Mini trained on ChEMBL than the corresponding baseline model, levels out instead of decreasing, resulting in a higher final novelty than for the corresponding baseline model. Novelty for the Decoder Mini trained on PubChem does, however, reaches a higher value throughout all epochs compared to the novelty of the baseline decoder. The validity and uniqueness of the Decoder Mini never reach as high values as the baseline decoder.



(a) Decoder (ChEMBL)

(b) Decoder (PubChem)

Figure 4.9: Validity, uniqueness, and novelty over epochs for smaller decoder models with corresponding baseline decoder results in dashed lines. The trends in uniqueness and validity across epochs for the smaller decoder closely resemble those observed in the baseline decoders. This, while the Decoder Mini models exhibit higher novelty scores than their respective baseline decoders.

4.3 Reinforcement Learning Optimization

In the following experiments, reinforcement learning was employed to optimize the generation of new molecules using the pretrained models described in previous sections. The purpose of applying reinforcement learning is to guide the pretrained models toward generating molecules with desirable properties, as defined by the scoring functions. In this approach, reinforcement learning serves primarily as an enhanced sampling method rather than as a finetuning procedure, making all compounds generated throughout the reinforcement learning run equally valuable.

The models were optimized for the five different optimization tasks and corresponding scoring functions described in section 3.4.1 to assess two key aspects: the ability of each model to adapt to a new task and the diversity among the generated compounds. For every RL task, QED (Quantitative Estimate of Drug-likeness) was also included as an additional scoring criterion to maintain a drug-relevant bias. The total score for each compound is the geometric mean, with unit weight, over the intended scoring function and the QED score.

We begin by presenting the results from the simplest optimization task: to avoid generating compounds containing sulphur. This was only carried out for the baseline models, i.e., those without randomized SMILES during pretraining, presented in Section 4.2.1. While this task could have been replaced by simply filtering out all compounds containing sulphur, this simple task serves as proof of principle, demonstrating the feasibility of training implemented models using RL.

For all other tasks, the resulting scores are systematically compared across different models and datasets, comparing first the inter-architectural performance, i.e., comparing the different training conditions (ChEMBL versus PubChem and randomized SMILES versus canonical SMILES) for each model separately. We then compare the scores between the better-performing variants of each architecture against each other. The scores presented here are the moving averages over the training steps, as the average score per step fluctuates. Hence, the score presented at each step is the average over the scores of the 10 preceding and succeeding steps. The moving average is applied to both the task-specific score and the QED.

To study the diversity among the generated compounds, the number of unique compounds, scaffolds, and generic scaffolds over RL runs was computed, and these are compared across all models and training conditions. To display the relevance of these, the number of compounds and scaffolds that achieve a task score and QED above 0.6 is visualized together with the total number of compounds and scaffolds.

Below, we present the results from the RL tasks: *Sulphur Avoidance* (baseline), *Similarity Guided Structure Generation*, and *Target Activity: DRD2*, while the results from the remaining RL tasks: *Target Activity: JNK3*, *Target Activity: GSK3β*, and are shown in Appendix B.

4.3.1 Sulphur Avoidance

As a proof of principle, the models were trained to avoid generating molecules containing sulphur. To achieve this, the scoring component described by Equation 3.1 was utilized, along with a QED scoring component. The geometric mean of these scores (with unit weights) was used to train the models. Achieving an average score of 1 indicates complete sulphur avoidance, and high QED values suggest that the generated molecules possess drug-like attributes. In Figure 4.10, the per-step average score and QED can be seen for the decoder, RNN, and Mamba pretrained on ChEMBL and PubChem (with canonical SMILES during pre-training).

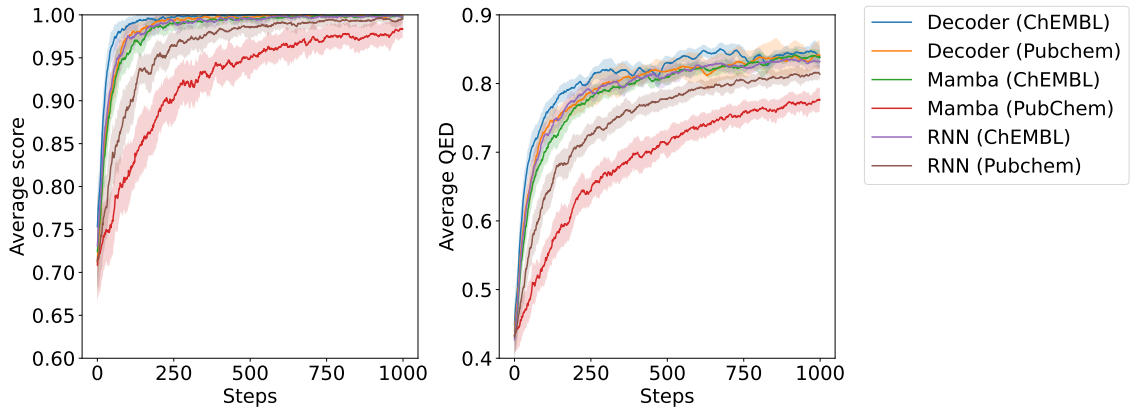


Figure 4.10: Per step average score and QED over number of steps for the decoder and RNN pretrained on ChEMBL and PubChem (as indicated by the legend to the right), over an RL run with sulphur avoidance and QED as scoring components. Each line indicates an average over three consecutive RL runs, and the shaded area is the average ± the standard deviation over the three runs.

All models except for the RNN and Mamba pretrained on PubChem reach an average score of 1 within the first few hundred epochs, as well as an average QED value of about 0.8. The RNN pretrained on PubChem reaches an average score of 1 much later, while Mamba pretrained on PubChem never reaches an average score of 1. For the average QED, the RNN pretrained on PubChem reaches ~ 0.8 in the later steps, while Mamba pretrained on PubChem does not reach 0.8.

In addition to the scorings, the sampled SMILES during the RL run were recorded, and their Murcko scaffold as well as the generic Murcko scaffold were computed. In Figure 4.11, the average number of unique compounds, Murcko scaffolds, and generic Murcko scaffolds with a score and QED above 0.6 can be seen. Being that a total of 64.000 compounds are sampled over one RL run, a large fraction of those compounds are unique for all models. However, less than half of those compounds are sulphur-free (score = 1) and have a QED above 0.6 for the decoder and RNN models.

As shown in Figure 4.11a and 4.11b, Mamba pretrained on both ChEMBL and PubChem produces the highest number of unique compounds and scaffolds while

also having a QED above 0.6. All the other models perform similarly in producing unique compounds, scaffolds, and generic scaffolds.

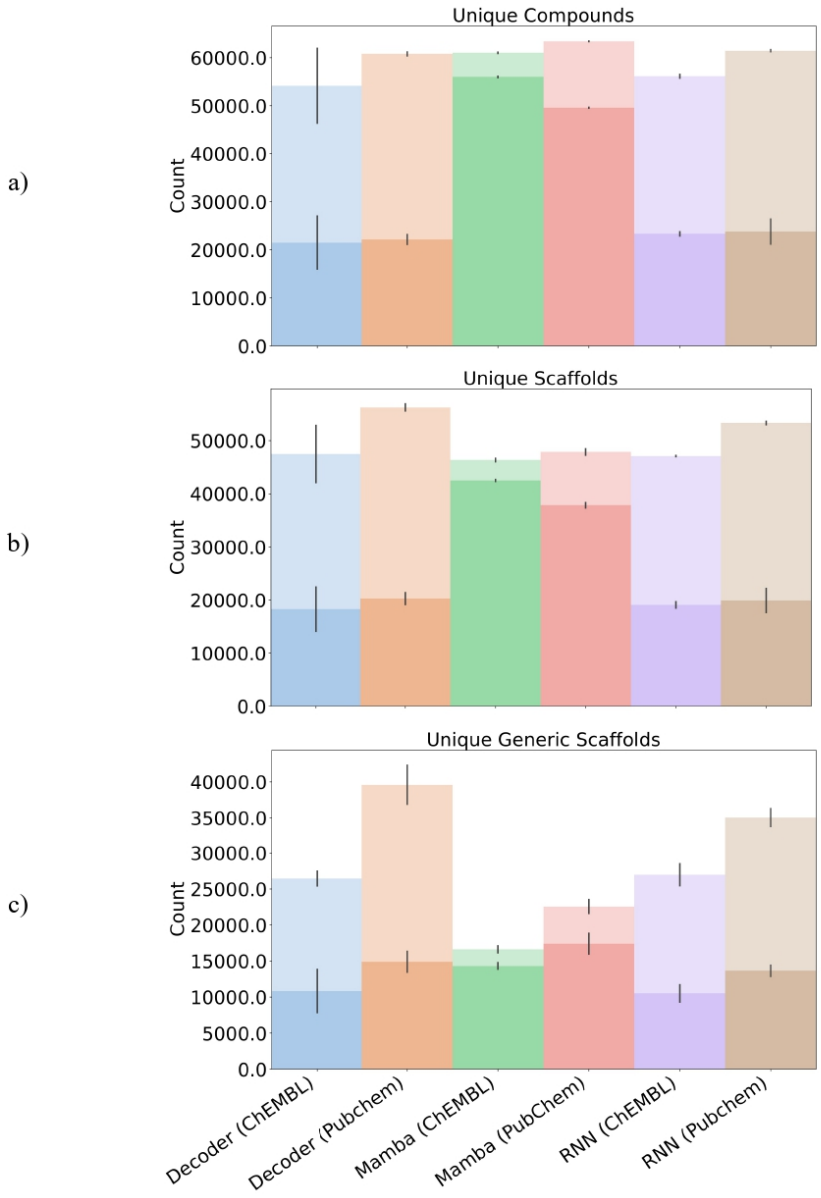


Figure 4.11: Number of (a) unique compounds, (b) unique scaffolds, and (c) unique generic scaffolds generated by each model over an RL run of 1000 steps with a batch size of 64. The total number of compounds/scaffolds is illustrated by the transparent bar, and the solid bar represents the number of compounds/scaffolds that also have a score and QED over 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these three runs).

4.3.2 Similarity Guided Structure Generation

Measuring the similarity of the generated molecules to the target Celecoxib, the Tanimoto similarity was used as explained in Section 3.4.1. Scoring for QED was added and the total score used for training was the geometric mean over the similarity score and the QED score, with unit weights. In contrast to the other experiments, the diversity filter described in Section 3.4.2 was not used. Instead, a filter to penalize repeatedly generated SMILES was used. Every recurring SMILES was penalized by multiplying the score by 0.5. This was done since the purpose of the task is for the model to generate molecules within a narrow chemical space, rather than exploring a broad space to find different kinds of compounds that fulfill some criteria. The per-step average similarity score and QED can be seen in Figure 4.12 (RNN), 4.13 (Decoder), and 4.14 (Mamba). Compared to the results seen for the sulphur avoidance task, all models achieve a rather low similarity score on average. However, the average similarity score increases over the training steps for all architectures.

The RNN models, whose scores can be seen in Figure 4.12, have smooth similarity scores and some more oscillations in the QED score. The ChEMBL pretrained RNNs achieve higher both similarity and QED score throughout the RL run, than the RNNs pretrained on PubChem. Comparing the RNNs pretrained with and without randomized SMILES during pre-training, there is little to no difference for those pretrained on ChEMBL. In contrast, the RNNs pretrained on PubChem with randomized SMILES have lower similarity and QED scores over most of the RL runs.

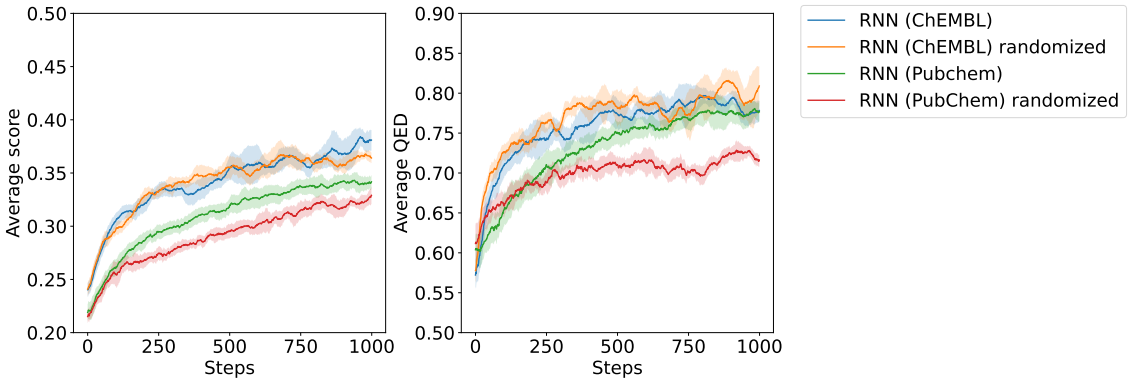


Figure 4.12: RNNs average score and QED over steps in RL task: Similarity guided structure generation (Celecoxib). The RNNs pretrained on ChEMBL achieve higher similarity and QED scores throughout the RL run compared to those pretrained on PubChem. The RNNs pretrained on PubChem with randomized smiles achieve the lowest average score and QED. The shaded area is the average ± the standard deviation over three runs.

Similarly, for the decoder models, the models pretrained on ChEMBL achieve higher similarity scores over the RL run. The similarity and QED scores, however, are not as smooth for the decoder models, and fluctuations in both scores are evident for all decoder models in Figure 4.13. While the decoder model pretrained on PubChem

with randomized SMILES has the lowest similarity score over the run, the other PubChem-pretrained model reaches a similarity score comparable to that of the CHEMBL-pretrained models by the end of the RL run. In addition to this, the decoder pretrained with PubChem and non-randomized SMILES maintains a QED score similar to those of the ChEMBL-pretrained decoders after about 500 steps.

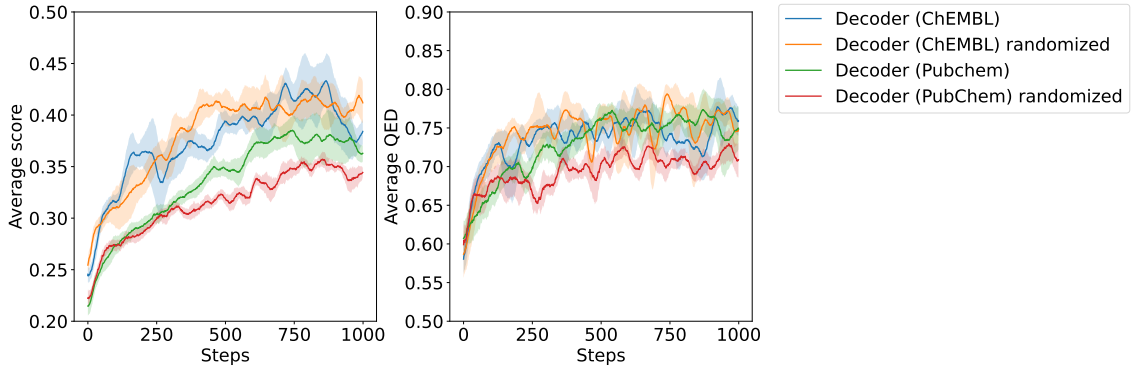


Figure 4.13: Decoders average score and QED over steps in RL task: Similarity guided structure generation (Celecoxib). The decoders pretrained on ChEMBL achieve higher similarity scores throughout the RL run compared to those pretrained on PubChem. For both the average score and QED, the decoder pretrained on PubChem with randomized smiles achieves the lowest while the decoder pretrained on PubChem without randomized smiles reaches similar levels as the decoders pretrained on ChEMBL. The shaded area is the average ± the standard deviation over three runs.

For the Mamba models, whose scores can be seen in Figure 4.14, the ChEMBL-pretrained models achieve higher similarity and QED scores throughout the RL runs, and the distinction between these and the PubChem-pretrained models is larger than for the RNN and the decoder models. The effect of pretraining with randomized SMILES also differs for the Mamba models compared to the RNN and the decoder: the model pretrained on ChEMBL with randomized SMILES achieves a distinctly higher similarity score over the RL run than the ChEMBL-pretrained model without randomized SMILES. For the PubChem-pretrained Mamba models, however, there is little to no difference in similarity and QED score when training with versus when training without randomized SMILES.

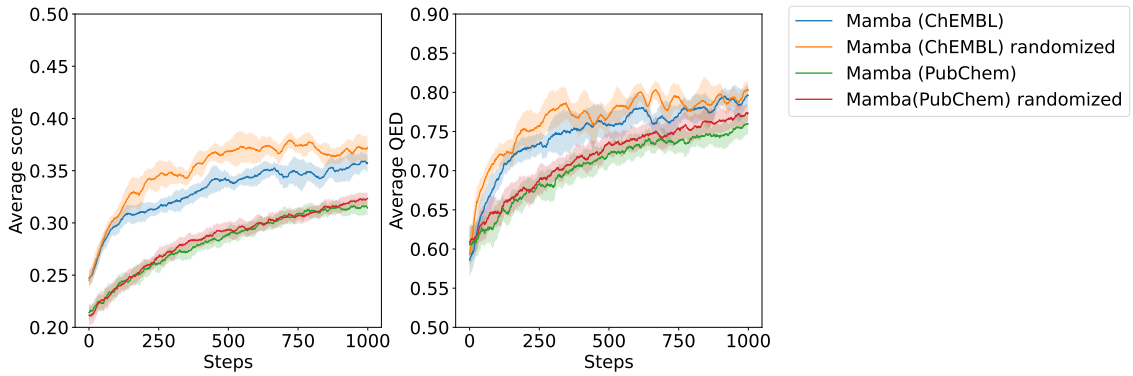


Figure 4.14: Mamba average score and QED over steps in RL task: Similarity guided structure generation (Celecoxib). The models pretrained on ChEMBL achieve higher scores than those pretrained on PubChem throughout the RL run. The model pretrained on ChEMBL achieves a higher average QED early in the RL run but performs similarly to the models pretrained on PubChem later on during the run. The shaded area is the average ± the standard deviation over three runs.

To compare the performance between the three different architectures, the architectures pretrained with ChEMBL and randomized SMILES were chosen for comparison, as these pretraining conditions resulted in high similarity and QED scores for all three architectures. The similarity and QED scores over the RL runs for the decoder, RNN, and Mamba pretrained with ChEMBL and randomized SMILES can be seen in Figure 4.15. While the performance of the three different architectures is similar, the Decoder achieves a slightly higher similarity score than the other two architectures. On the other hand, the fluctuations in QED score are much larger for the decoder model, whereas the other two architectures have steadier QED scores.

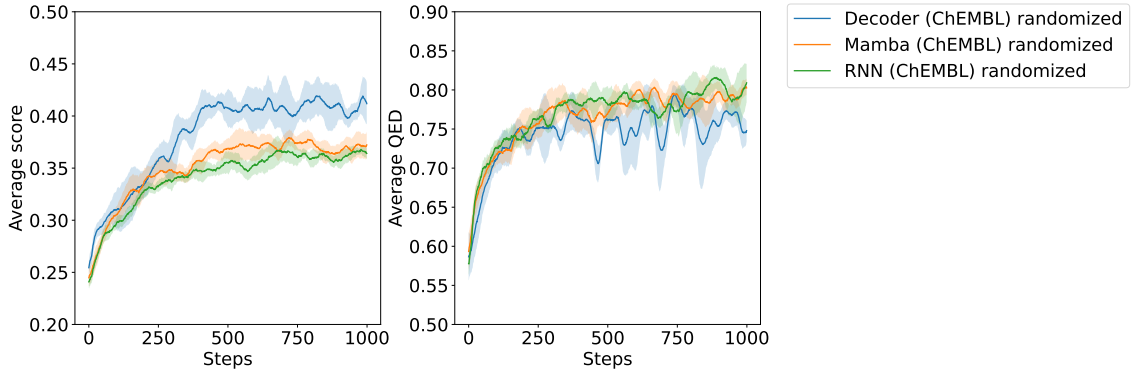


Figure 4.15: Decoder, RNN, and Mamba (Randomized smiles) average score and QED over steps in RL task: Similarity guided structure generation (Celecoxib). The decoder has a higher average score during the RL run, but has larger fluctuations in QED compared to the RNN and Mamba, which have comparably steady QED over the RL run. The shaded area is the average ± the standard deviation over three runs.

When optimizing the models through RL to generate molecules similar to the query

compound, the aim is to narrow the chemical space of generated molecules while maintaining a variety among generated samples. In Figure 4.16, the number of unique compounds generated over the RL runs can be seen for each model. While all models did increase their average scores over the RL runs, the highest averages were around 0.4. Hence, when studying the number of unique compounds, very few of them reach a similarity score (and QED score) above 0.6. However, the number of unique compounds overall is high for all models and pretraining configurations, reaching above 50.000 unique compounds. Comparing the different pretraining configurations, all models exhibit the same pattern: models pretrained on PubChem generate more unique compounds than those trained on ChEMBL, and pretraining with randomized SMILES also results in more unique compounds being generated. Looking at the number of unique scaffolds as seen in Figure 4.17, two models stand out: the decoder and RNN pretrained on PubChem with randomized SMILES generate significantly more unique scaffolds than the other models. On the other hand, corresponding models trained without randomized SMILES generate the most generic scaffolds, which is evident from Figure 4.18. The decoder and RNN pretrained on PubChem without randomized SMILES both generate around 20.000 unique generic compounds, which is a large fraction of their unique scaffolds (which amount to approximately 25.000 compounds). This is, while corresponding models pretrained with randomized SMILES generated the most unique scaffolds, the fraction of those being generically unique is much lower.

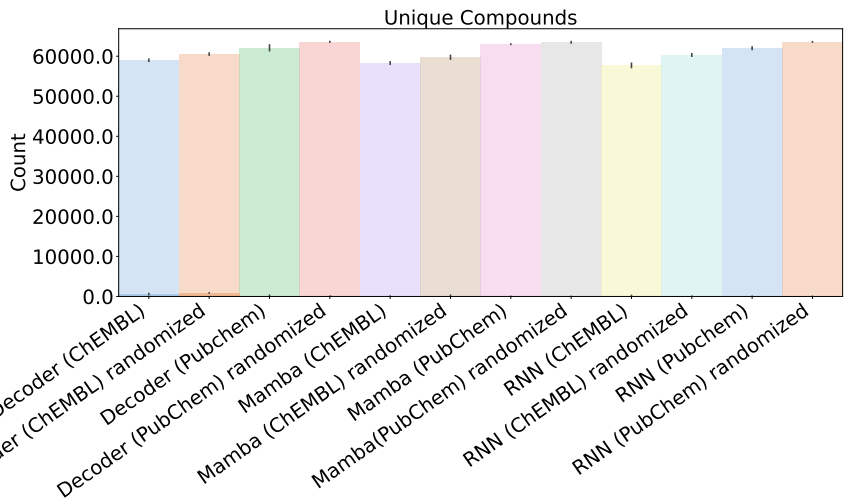


Figure 4.16: Number of unique compounds generated by all the models pretrained on both datasets with and without randomized SMILES over 1000 steps with a batch size of 64 in RL task: Similarity guided structure generation (Celecoxib). The total number of compounds is illustrated by the transparent bar, and the solid bar (only visible for Decoder (ChEMBL) randomized) represents the number of compounds that also have a score and QED over 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these three runs).

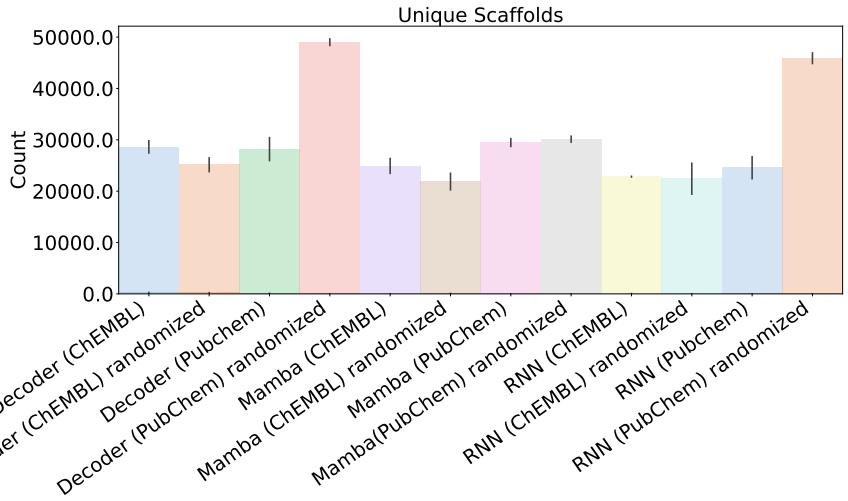


Figure 4.17: Number of unique scaffolds generated by all the models pretrained on both datasets with and without randomized SMILES over 1000 steps with a batch size of 64 in RL task: Similarity guided structure generation (Celecoxib). The total number of scaffolds is illustrated by the transparent bar, and the solid bar (not visible for any of the models) represents the number of scaffolds that also have a score and QED over 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these three runs).

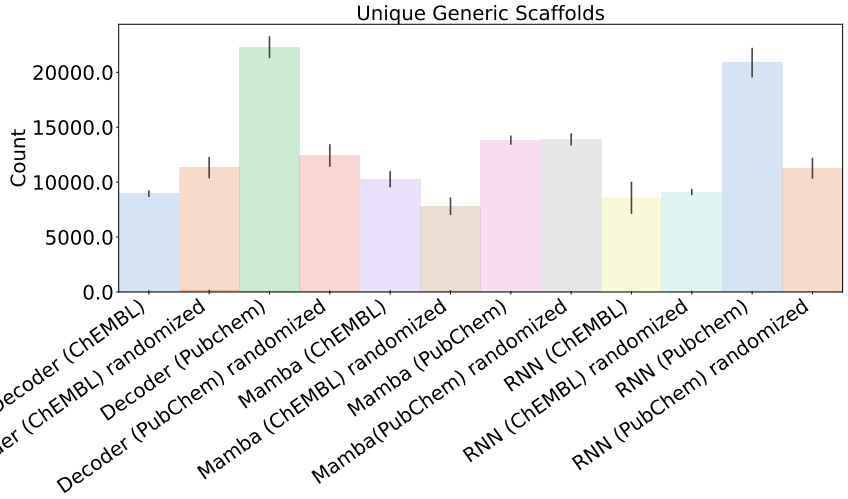


Figure 4.18: Number of unique generic scaffolds generated by all the models pretrained on both datasets with and without randomized SMILES over 1000 steps with a batch size of 64 in RL task: Similarity guided structure generation (Celecoxib). The total number of generic scaffolds is illustrated by the transparent bar, and the solid bar (not visible for any of the models) represents the number of generic scaffolds that also have a score and QED over 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these three runs).

Among the generated molecules during one of the RL runs, the unique compounds

with the highest similarity score can be seen in Figure 4.19 (canonical pretraining) and 4.20 (randomized pretraining). Many of the models do on some occasions generate the query compound, however these have been excluded from the figures.

The Celecoxib molecule has a core structure consisting of three rings: two phenyl rings (six-membered carbon ring) and one Pyrazole ring (five-membered with 3 carbon and two nitrogen). These three rings can be seen in a majority of the compounds seen in both Figure 4.19 and 4.20. Another feature of the Celecoxib molecule is the sulfoanamide group attached to one of the phenyl rings (to the top right in red, yellow, and blue in the target structure at the top of the figures). This as well is found in many of the compounds in Figure 4.19. However, parts of it do sometimes occur between the two phenyl rings, as seen for the two highest-scoring compounds generated by the decoder pretrained on PubChem with randomized SMILES. The trifluoromethyl group (in light blue to the top left in the target structure at the top of the figures), which is another key feature of the Celecoxib molecule, is also evident in many of the high-scoring compounds.

To provide deeper insight into how the overall similarity to the query compound changes with RL, the similarity between Celcoxib and the first 1000 and final 1000 generated compounds was computed. The distribution of these similarities for each of the models can be seen in Figure 4.21. All models do successfully shift the distribution towards a higher similarity to Celecoxib. The models pretrained on PubChem (see Figure 4.21b, 4.21d) have wider and flatter distributions for the first 1000 compounds. However, for the final 1000 compounds, the distributions have narrowed, with a more pronounced peak, similar to the corresponding distribution for the ChEMBL trained models (see Figure 4.21a, 4.21c).

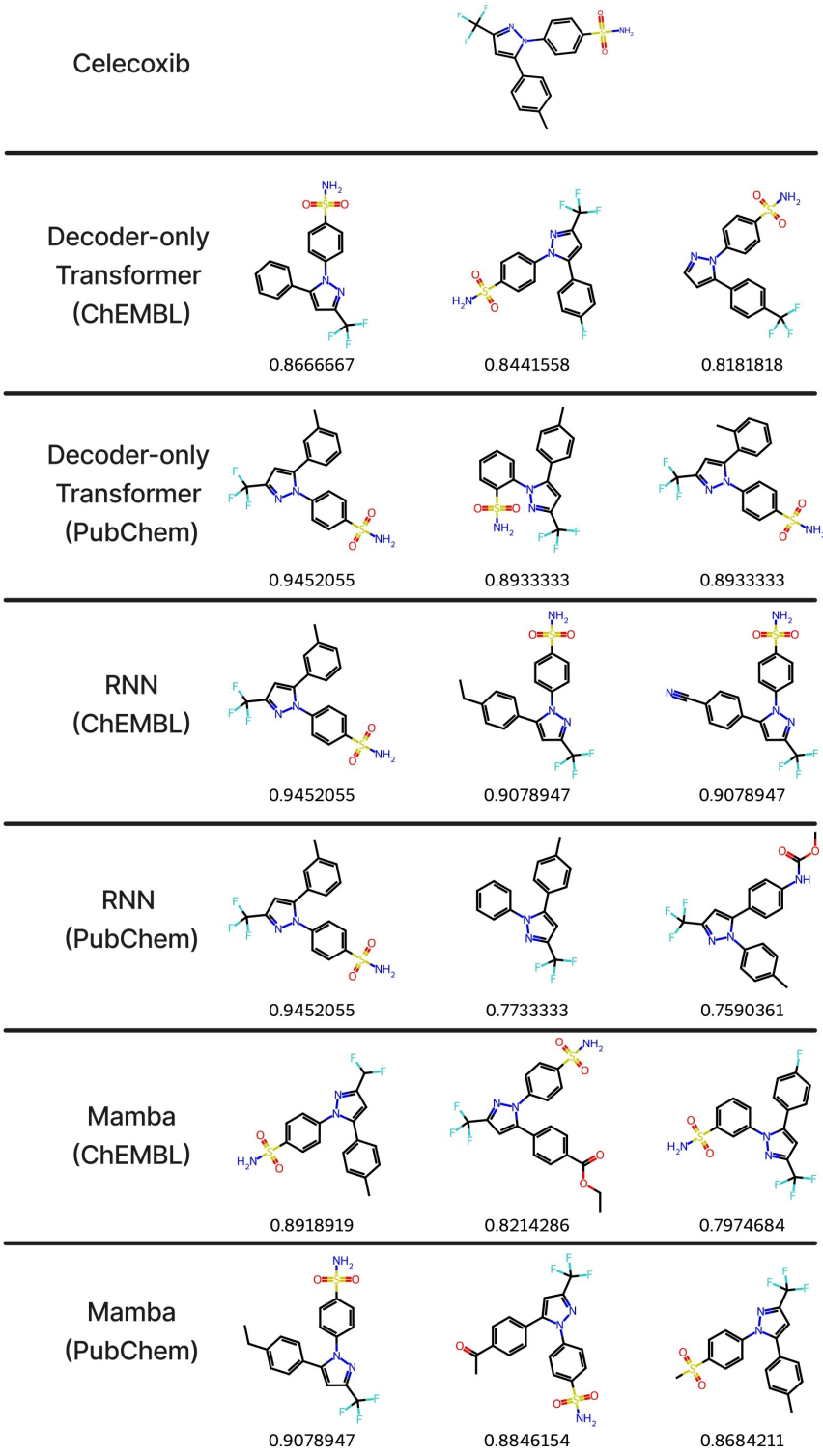


Figure 4.19: Sampled molecules after RL for similarity guided structure generation and the query compound, Celecoxib. The value underneath each molecule is the similarity score (Tanimoto) to Celecoxib.

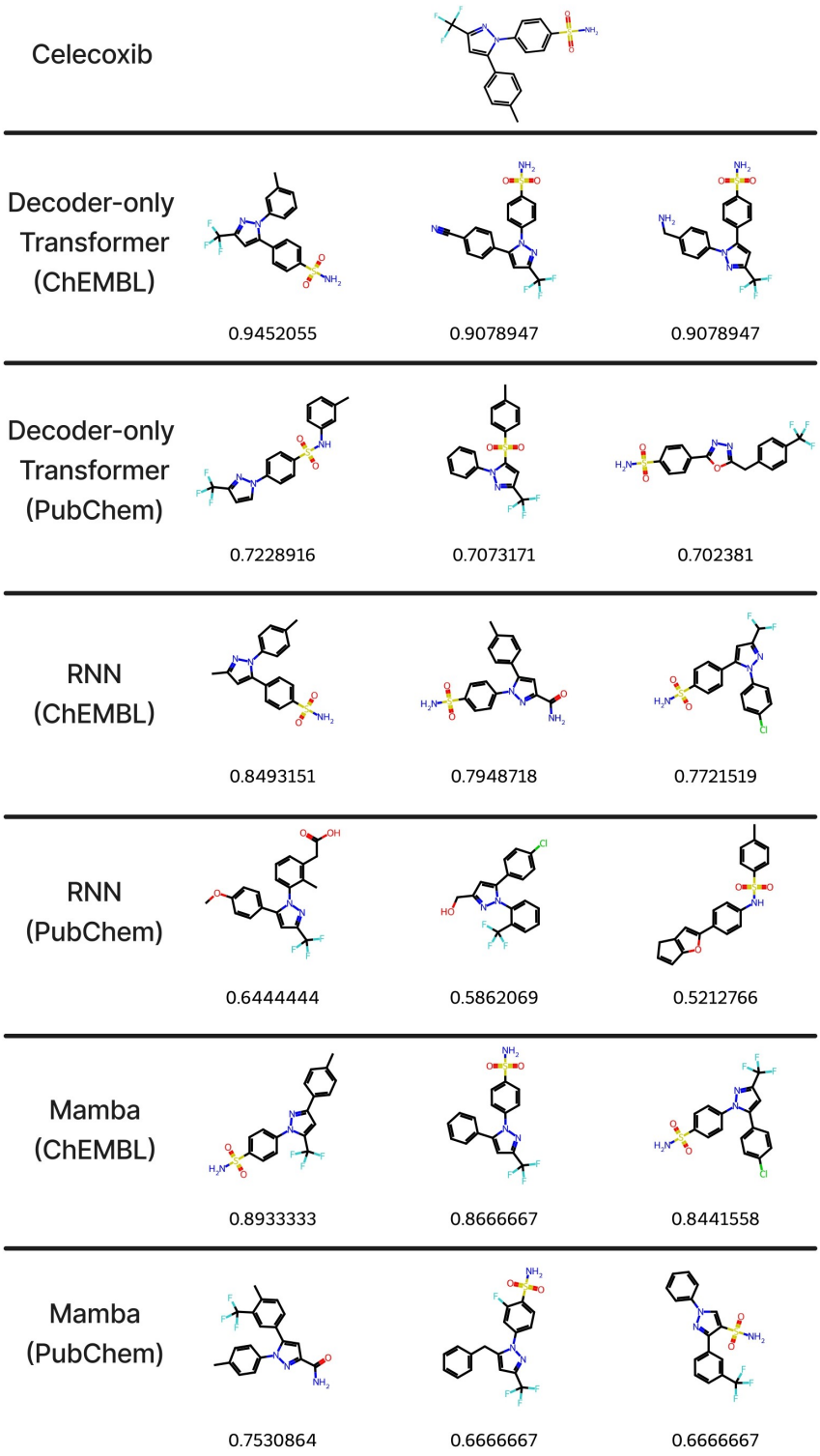
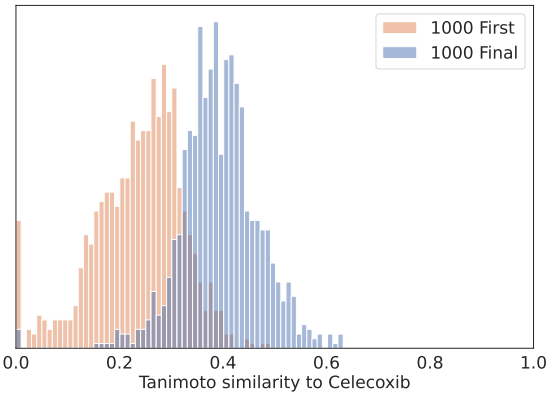
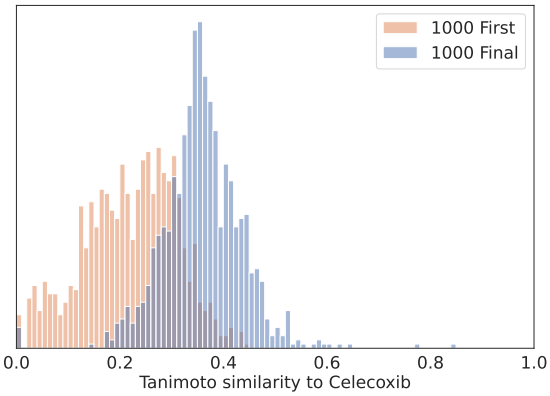


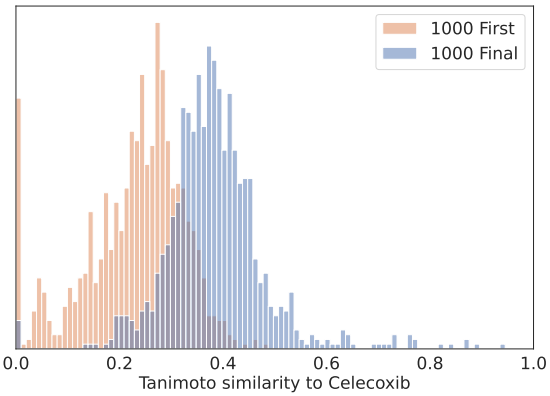
Figure 4.20: Sampled molecules after RL for similarity guided structure generation and the query compound, Celecoxib, with models trained on randomized SMILES. The value underneath each molecule is the similarity score (Tanimoto) to Celecoxib.



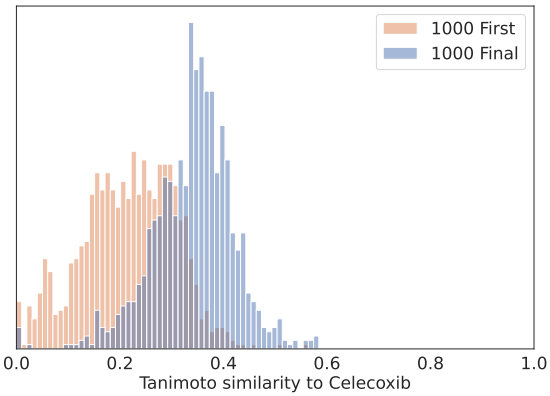
(a) Decoder (ChEMBL)



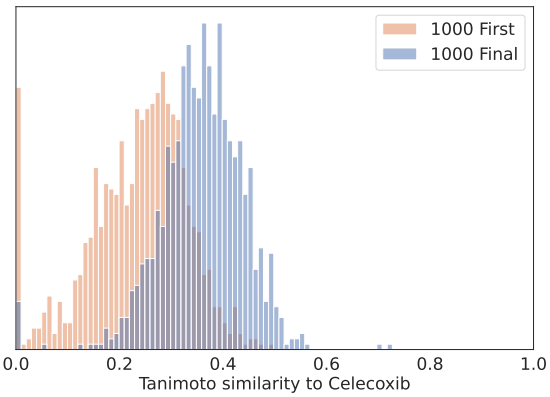
(b) Decoder (PubChem)



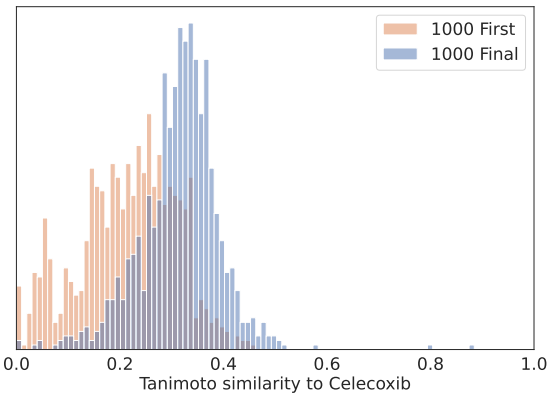
(c) RNN (ChEMBL)



(d) RNN (PubChem)



(e) Mamba (ChEMBL)



(f) Mamba (PubChem)

Figure 4.21: Similarities between the first 1000 sampled compounds (orange) and the query compound, Celecoxib, and between the final 1000 sampled compounds (blue) and the query compound. All models do successfully shift the distribution towards a higher similarity to Celecoxib. The RNN pretrained on ChEMBL produces a few compounds with very high similarity scores in the final sampling, more than the other models.

4.3.3 DRD2

To compare how well the models adapted to RL optimization towards activity against the dopamine type 2 receptor (DRD2), a support vector machine classifier was used to predict a generated compound's DRD2 activity as described in Section 3.4. Firstly, the impact from the dataset was evaluated on each of the models, and finally, an evaluation between the three models was made using the model from each architecture that was pretrained on ChEMBL using randomized SMILES. In Figure 4.22, the average score and QED over steps in RL for target activity DRD2 for the RNN across datasets are shown.

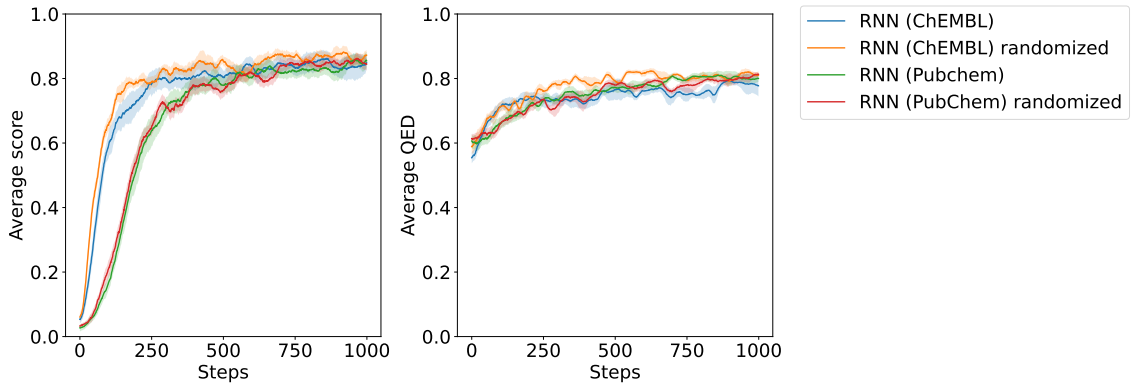


Figure 4.22: RNN average score and QED over steps in RL for target activity (DRD2) across datasets. The models pretrained on ChEMBL adapts faster to the task as compared to the models pretrained on PubChem as seen on the average score converging around 0.8. All models adapt similarly to the QED also around 0.8. The shaded area is the average ± the standard deviation over three runs.

The RNN pretrained on ChEMBL adapts relatively fast to the task, reaching average scores close to convergence after 250 steps. Those pretrained on PubChem reach those scores after around 500 steps.

In Figure 4.23, the average score and QED over steps in RL for target activity DRD2 for the decoder across datasets is shown. Similarly to the RNN, the decoders pretrained on ChEMBL learns the task faster, but this difference is not as prominent as for the RNN as per Figure 4.22.

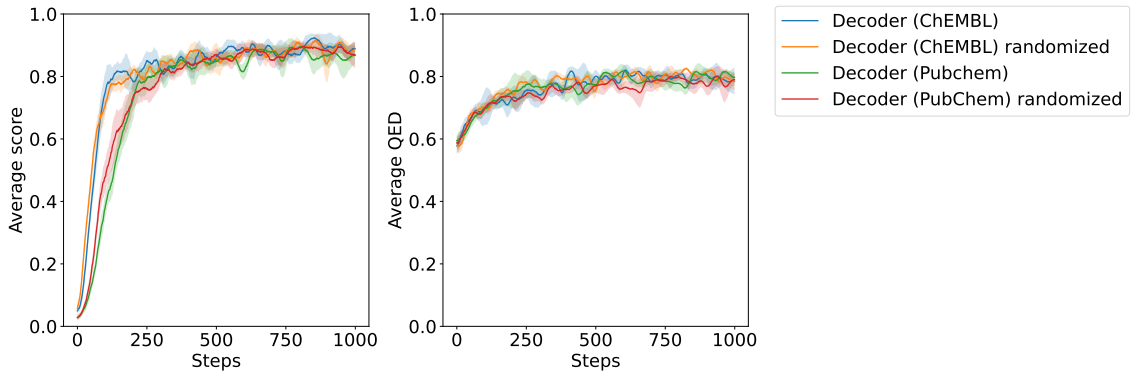


Figure 4.23: Decoder average score and QED over steps in RL for target activity (DRD2) across datasets. The models pretrained on ChEMBL adapt faster to the task as compared to the models pretrained on PubChem, as seen on the average score converging slightly above 0.8. All models adapt similarly to the QED around 0.8. The shaded area is the average ± the standard deviation over three runs.

In Figure 4.24, the average score and QED over steps in RL for target activity DRD2 for the Mamba across datasets are shown. The Mamba models show greater variance in adapting to the task according to the average score. As for the other model architectures, the models pretrained on ChEMBL learns the task during fewer steps than those pretrained on PubChem.

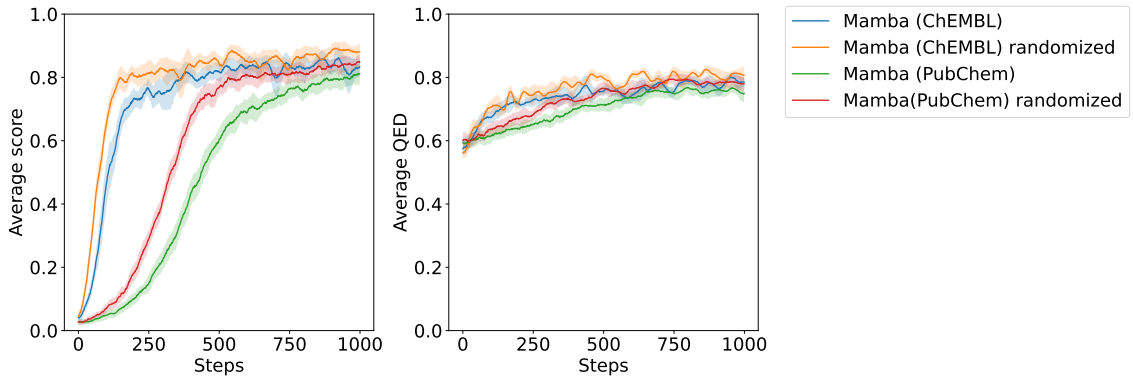


Figure 4.24: Mamba average score and QED over steps in RL for target activity (DRD2) across datasets. The models pretrained on ChEMBL adapt faster to the task as compared to the models pretrained on PubChem, as seen on the average score converging around 0.8. All models adapt similarly to the QED around 0.8. The shaded area is the average ± the standard deviation over three runs.

In Figure 4.25, the average score and QED over steps in RL for target activity DRD2 for the decoder, RNN, and Mamba pretrained on ChEMBL randomized SMILES are shown. When comparing the models pretrained on the same dataset (ChEMBL randomized SMILES), the difference in adapting to the task based on average score and QED is minuscule.

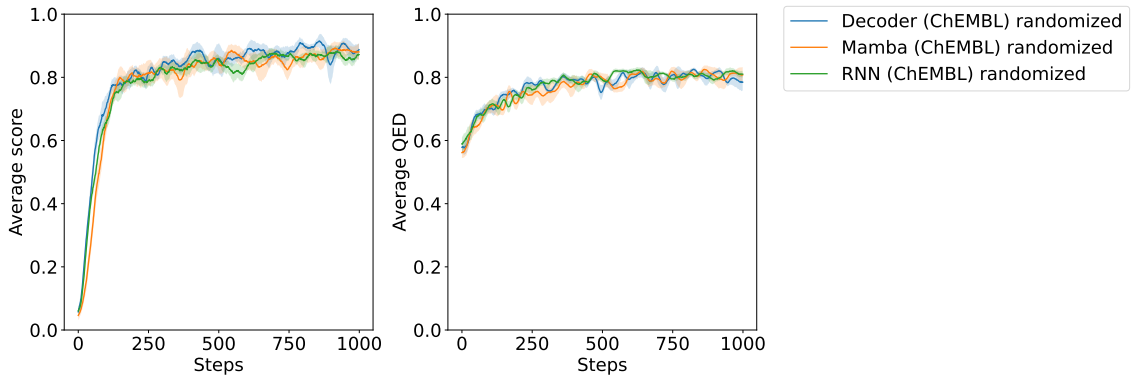


Figure 4.25: Decoder, RNN, and Mamba average score and QED over steps in RL for target activity (DRD2) pretrained on ChEMBL randomized SMILES. The difference in average score and QED is minuscule across the model architectures, all converging at an average score of 0.8, reaching a QED of 0.8 at a similar pace. The shaded area is the average ± the standard deviation over three runs.

When evaluating the number of unique compounds over the 1000 steps per model-dataset pair, most models generate similar amounts of unique compounds, all generating around 60.000 as shown in Figure 4.26. When including a threshold of a compound also having a score and QED > 0.6, the decoder exhibits a more robust behavior, producing similar amounts of unique compounds across the datasets, while the RNN and Mamba exhibit varying performance.

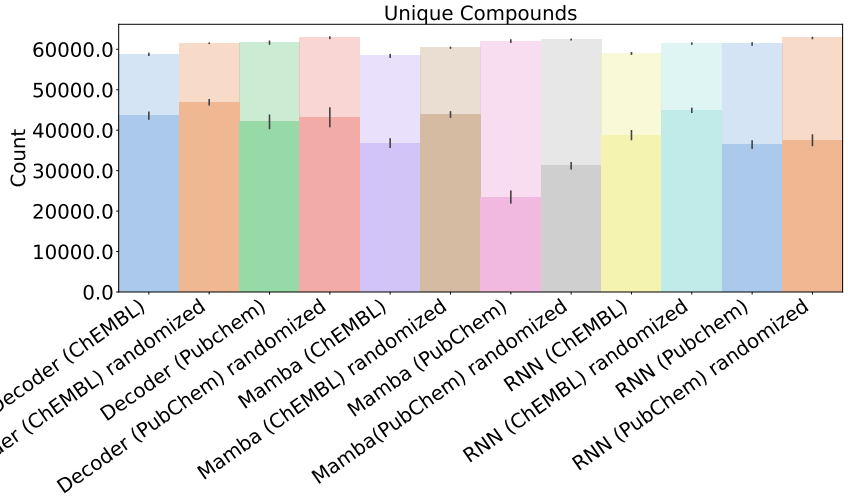


Figure 4.26: Number of unique compounds generated by all the models pretrained on both datasets with and without randomized SMILES over 1000 steps with a batch size of 64 in RL for target activity (DRD2). The total number of compounds is illustrated by the transparent bar, and the solid bar represents the number of compounds that also have a score and QED over 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these three runs).

Figure 4.27 shows the number of unique scaffolds over the 1000 steps per model-dataset pair. The amount of unique scaffolds varies between the models. All models produce the most unique scaffolds when pretrained on PubChem randomized SMILES and the fewest when pretrained on ChEMBL without randomized SMILES. When including a threshold of a compound also having a score and QED > 0.6, the decoder exhibits more robust behavior than the other models, producing similar amounts of unique scaffolds across model-dataset pairs.

Figure 4.28 shows the number of unique generic scaffolds over the 1000 steps per model-dataset pair. The amount of unique generic scaffolds varies between the models. The amount of generated unique generic scaffold behaves similarly between the datasets for the decoder and the RNN, but Mamba shows another behavior. Mamba, pretrained on PubChem randomized SMILES, produces the most unique generic scaffolds, while the decoder generates the most when pretrained on ChEMBL without randomized SMILES. For the RNN, being pretrained on PubChem without randomized SMILES generates the most unique generic scaffolds. When including a threshold of a compound also having a score and QED > 0.6, the decoder generates the most unique generic scaffolds overall.

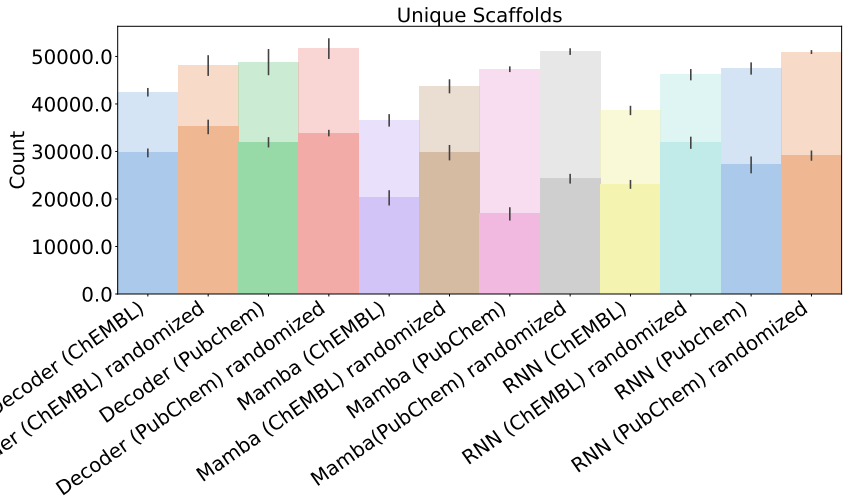


Figure 4.27: Number of unique scaffolds generated by all the models pretrained on both datasets with and without randomized SMILES over 1000 steps with a batch size of 64 in RL for target activity (DRD2). The total number of scaffolds is illustrated by the transparent bar, and the solid bar represents the number of scaffolds that also have a score and QED over 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these three runs).

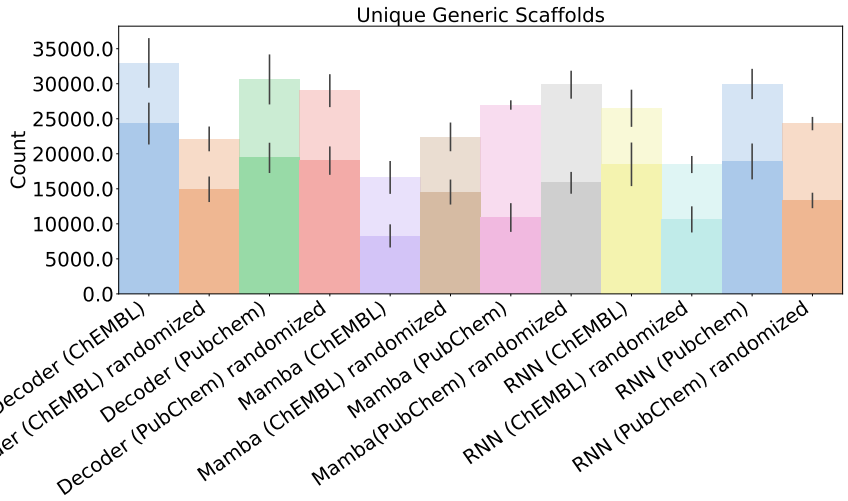


Figure 4.28: Number of unique generic scaffolds generated by all the models pretrained on both datasets with and without randomized SMILES over 1000 steps with a batch size of 64 in RL for target activity (DRD2). The total number of scaffolds is illustrated by the transparent bar, and the solid bar represents the number of scaffolds that also have a score and QED over 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these three runs).

4.3.4 Decoder Mini and DRD2 activity

The Decoder Mini, described in Section 4.2.4, was also trained using reinforcement learning for various optimization tasks. For target activity toward DRD2, the evolution of the activity score and QED across epochs for the Decoder Mini trained on ChEMBL (without randomized SMILES) is displayed in Figure 4.29, alongside the corresponding scores for both the baseline decoder and the RNN trained on ChEMBL. The Decoder Mini demonstrates comparable performance to the baseline decoder with respect to both activity and QED scores throughout the training epochs. Notably, both models marginally outperform the RNN on these metrics.

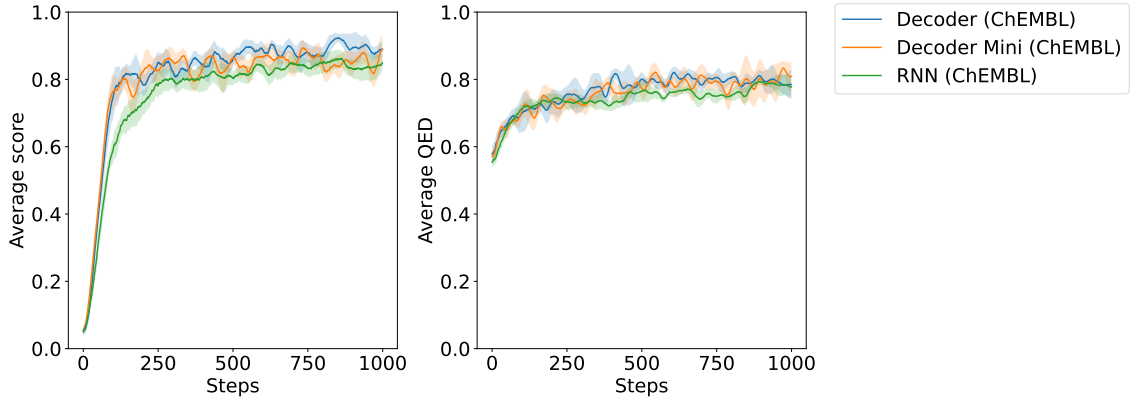
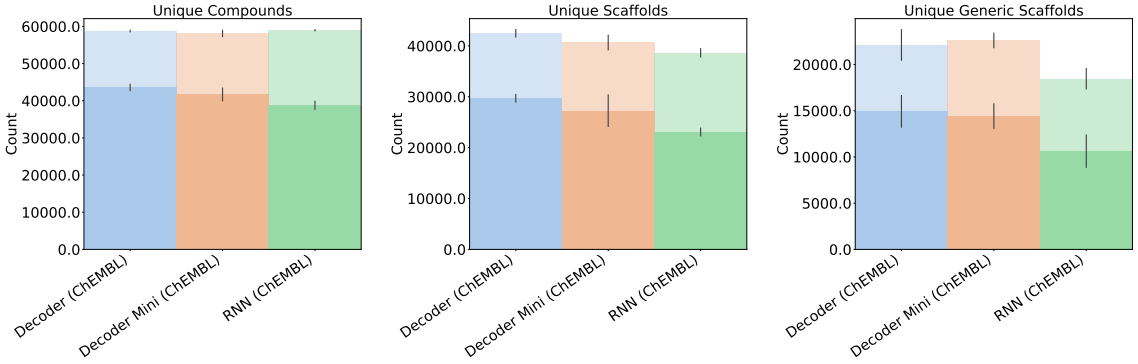


Figure 4.29: Per step average target activity (DRD2) and QED score over steps for the baseline decoder, Decoder Mini, and RNN pretrained on ChEMBL (as indicated by legend to the right), over an RL run with DRD2 activity and QED as scoring components. Each line indicates an average over three consecutive RL runs, and the shaded area is the average ± the standard deviation over the three runs.

Figure 4.30 illustrates the number of unique compounds, scaffolds, and generic scaffolds generated by the Decoder Mini, the baseline decoder, and the RNN during reinforcement learning for DRD2 activity. For all three of these diversity measures, Decoder Mini performs comparably to both the baseline decoder and RNN, consistently achieving slightly higher scores than the RNN across all metrics.

The results of training the Decoder Mini for other optimization tasks can be found in Appendix B.4.



(a) (b) (c)

Figure 4.30: Number of (a) unique compounds, (b) unique scaffolds, and (c) unique generic scaffolds generated by each model over an RL run of 1000 steps with a batch size of 64 for target activity (DRD2). The total number of compounds/scaffolds is illustrated by the transparent bar, and the solid bar represents the number of compounds/scaffolds that also have a score and QED over 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these three runs).

5

Discussion

This discussion has been structured around the research questions of this thesis. Rather than providing an exhaustive evaluation, this discussion selectively highlights those observations and trends we find most relevant. Throughout, we aim to assess the strengths and limitations of the undertaken methods, examining the impact of both methodological choices and underlying model dynamics.

5.1 The Choice of Architecture

During pretraining, some distinct learning dynamics became apparent, with both the Mamba and RNN reaching their minimum validation losses considerably faster than the decoder when trained on ChEMBL, as seen in Figure 4.4. This more rapid convergence for Mamba and RNN may reflect architectural efficiencies in sequence modeling, permitting faster adaptation to the structure of molecular data. While these learning dynamics are clear for the ChEMBL-trained models, none of the PubChem-trained models were trained long enough for those dynamics to be observed. Hence, it is impossible to tell whether these dynamics generalize to larger datasets. The seeming fact that the RNN and Mamba tend to overfit to ChEMBL may be connected to a combination of factors. Model size relative to dataset size, architectural properties, or suboptimal hyperparameter tuning. Given that ChEMBL is a bioactive-focused dataset and considerably smaller than PubChem, it may be insufficiently large for the scale and complexity of the RNN and Mamba. The ratio of model size to dataset size is a well-documented cause of overfitting in that excessively complex models tend to overfit on less complex tasks and smaller datasets [57]. To ascertain whether this phenomenon has affected our experiments, a more detailed investigation into the relationship between model size and performance for both the RNN and Mamba architectures would be required. But prior to this, thorough hyperparameter tuning should be performed individually for each model.

While the validation loss offers insight into the generalization error of a model, it may not fully account for the degree to which the model has genuinely learned the underlying distribution of the data rather than simply memorizing the syntax of the data. The novelty metric, which was computed as the proportion of generated molecules not present in the training data, serves as a complementary metric on potential overfitting. High novelty in combination with high validity and uniqueness indicates that the model is exploring chemical space beyond what it has seen

during training, while a lower novelty would imply that the model is reproducing the training data rather than learning the underlying distribution of it. Studying the novelty curves of the models trained on canonical SMILES in Figure 4.5 with this in mind, it becomes evident that all three architectures, to some extent, memorize the training data long before reaching their minimum validation loss. On the other hand, this does not imply that training after the point of maximum novelty is purely harmful; all architectures continue to increase in validity and uniqueness. Ultimately, memorizing the data does seem inevitable when training with canonical SMILES if the models are to learn the chemical space. Comparing across architectures, all models trained with canonical SMILES see a decrease in novelty from early on in the training process, and while Mamba has the highest novelty at its lowest validation loss epoch, see Table 4.2, it also sees the steepest decrease in novelty over epochs.

While requiring more training time, see Table 4.1 for exact times, the decoder ultimately achieved the highest levels of validity and uniqueness across datasets. Across epochs, it also maintains the highest fraction of novel SMILES compared to other architectures, although the fast convergence of Mamba results in Mamba having the highest novelty at the point of lowest validation loss, see reported novelties in Table 4.2.

All three architectures demonstrated high capability in capturing the underlying chemical space of the datasets, as confirmed by analyses included in Appendix B.2. When molecules generated from each model were examined in terms of their distributions over key chemical descriptors, such as molecular weight and number of heavy atoms, similar distributions to those of the training data were found, supporting the notion that model pretraining yields comprehensive chemical coverage irrespective of architectural design.

Notably, performance differences were also apparent with respect to model scaling. The Decoder Mini variant of the decoder-only Transformer, with 2.2 M parameters instead of the 6.4 M parameters of the baseline decoder, performed as well as the substantially larger RNN (5.9 M parameters) regarding validity and uniqueness (Figures 4.9 4.5c, and 4.5d). In contrast to the larger models, the Decoder Mini's novelty kept increasing during training and reached its maximum novelty at the second to last epoch, showing signs of better generalization than other models. In addition to this, the Decoder Mini is significantly faster to train and sample than its larger counterpart. Based on novelty, the performance of the Decoder Mini strengthens the claim that less complex models are better at generalizing.

For the case of solely pretraining a model to encapsulate the underlying distribution of the training data, generating valid, unique, and novel molecules, the choice of the architecture's effect is limited. While the decoder-only Transformer exhibits distinct learning dynamics as it continues to improve its validation loss over extended epochs compared to the faster convergence of Mamba and RNN, this could in part be due to suboptimal hyperparameter choices for the latter models. The decoder consistently achieves slightly higher validity and uniqueness in its generated molecules, though these differences diminish when training on the larger PubChem dataset. Overall,

when the primary goal is to learn and reproduce the training data distribution during pretraining, the choice of architecture appears to have only a modest impact on the quality and diversity of generated molecules, especially as the dataset size increases.

5.2 The Choice of Training Data

ChEMBL and PubChem present contrasting opportunities and constraints for molecular generation. The most apparent difference between these datasets is scale, with ChEMBL comprising 2.2 M compounds and PubChem 119 M compounds, greatly expanding the accessible chemical space. While this increase in data volume does raise the expectation that PubChem could facilitate wider exploration and broader diversity, examination of the actual distributions of chemical attributes within each dataset shows that this diversity is not pronounced. Both datasets display similar spreads in key molecular descriptors, though important nuances appear upon closer inspection.

The distribution of QED of compounds seen in Figure 4.1 shows that ChEMBL's compounds tend to have a higher QED, with a substantially larger fraction exhibiting a QED above 0.5. This suggests that ChEMBL's curated nature favors medicinal relevance. In contrast, PubChem's distribution of heavy atom counts, seen in Figure 4.2, reveals a prevalence of smaller molecules clustered around twenty heavy atoms, whereas ChEMBL shows a pronounced peak at thirty. Examining ring count in Figure 4.3, PubChem is dominated by molecules with two rings, alongside a notable fraction with complex architectures comprising eight or more rings. ChEMBL preferentially contains compounds with three rings. The differences in these distributions between ChEMBL and PubChem are reproduced by all models, and a comparison between the distributions can be seen in Appendix B.2. Hence, the training data directly affects the chemical space accessible to the trained models.

The differences between ChEMBL and PubChem shape the behavior of the models during pretraining. Both the RNN and Mamba trained with ChEMBL overfitted after less than 20 epochs, as evident from the loss curves seen in Figure 4.4. The models trained on PubChem do not overfit, likely attributable to the immense dataset size. While this larger dataset makes it more difficult to overfit, the training times scale linearly with the number of samples in the dataset, causing the PubChem models to be slow to train. While the training time has not been an optimization objective, or of big importance to this thesis, it did cause them to be trained for fewer epochs due to computational and time limitations. Hence, it is possible that these models would eventually start overfitting.

The vast number of compounds in PubChem should make memorization of the training data more difficult or take a longer time, but despite this, models pretrained on PubChem consistently yield lower novelty by generating a smaller proportion of molecules not found in the training data. This might be due to the sheer size of PubChem, where, even though the model is not overfitting, it struggles to generate novel molecules since the dataset it has been trained on is so dense that many of the possible compounds within the space spanned by the dataset exist in the dataset.

In contrast, the models trained with ChEMBL and canonical SMILES (see Figure 4.5a, 4.5c, and 4.5e) show an early peak in novelty that then decreases at various speeds depending on the architecture. This implies that while the curated nature of ChEMBL makes it difficult for the models to memorize the data early in the training process, the small amount of compounds also makes it inevitable to memorize the training data as training proceeds.

Overall, PubChem-trained models exhibit a higher validity and uniqueness, both across epochs and at the epoch of lowest loss, than models trained with ChEMBL. The choice between a massive dataset like PubChem and a targeted, medicinally curated dataset such as ChEMBL must be weighed against the purpose of training the models, whether for broad chemical coverage, drug-likeness, or innovation. While the effects discussed here apply exclusively to the pretraining phase, the datasets' influence can be observed in downstream tasks such as reinforcement learning, with model performance and optimization outcomes shaped by the foundation laid by the training data.

5.3 The Effect of Randomized SMILES

The use of randomized SMILES during pretraining introduces distinct effects on the process of learning the underlying distributions of the training data. Across all architectures and datasets examined, the use of randomized SMILES resulted in significantly higher training losses compared to canonical SMILES, as seen in Figure 4.6. On the other hand, the general learning dynamics remain unchanged, with models following a similar progression in loss reduction and convergence to those seen when using canonical SMILES in Figure 4.4. Notably, both the Mamba and RNN architectures exhibit signs of early overfitting when trained on ChEMBL, regardless of SMILES representation.

While the validity and uniqueness of models trained with randomized SMILES seen in Figure 4.7 matches that of models trained with canonical SMILES (Figure 4.5), the impact on novelty is more pronounced. Models pretrained with randomized SMILES demonstrate substantially higher novelty across all datasets, with maximum achieved novelty values exceeding those obtained through canonical training. Within this regime, the decoder trained on ChEMBL achieves the highest novelty across architectures, whereas Mamba shows a slight advantage when training on PubChem. This improved novelty highlights one of the central aspects of randomizing SMILES: the removal of bias stemming from the canonicalization algorithm. As a result of this removal, models can not rely on systematic syntax, but instead are forced to learn the underlying chemical structure [58]. In this way, training with randomized SMILES helps models refrain from memorizing patterns in the training data before reaching their minimum validation loss, which is why we see the increasing novelty while loss is still decreasing (Figures 4.4 and 4.5).

The benefits of randomized SMILES during pretraining are expected to translate into reinforcement learning, where greater novelty and resistance to memorization can support more effective exploration of chemical space.

An alternative to exchanging each SMILES representation for a randomized SMILES of the same molecule would be to generate several randomized representations of the same molecule and include all of these in the dataset. This way, the models' generalization of the data could be influenced by an even more complex dataset, possibly impacting performance overall and the early overfitting of the RNN and Mamba models.

5.4 Reinforcement Learning Optimization

The sulphur avoidance task

The task of generating molecules without sulphur is relatively straightforward, as there is a direct correspondence between the scoring function and the explicit representation of sulphur tokens in the SMILES string. This stands in contrast to the other tasks evaluated, where the scoring function is determined by the molecular structure inferred from the SMILES, rather than the presence of specific tokens. Owing to this direct relationship, all models rapidly learn to avoid generating sulphur-containing compounds within the initial epochs of training, as seen in Figure 4.10. It is observed that the RNN and Mamba architectures pretrained on PubChem require more epochs to achieve an average score of 1 (and Mamba does not reach it at all), and while, for this task, generating low-scoring compounds is unfavorable, that does not have to be the case in general. While a high average score indicates that most generated compounds fulfil the criteria defined by the scoring function, the occasional generation of a lower-scoring compound suggests that these models retain a certain degree of exploration and are not exhibiting absolute mode collapse. This tendency toward exploration is further supported by the diversity metrics presented in Figure 4.11, with the Mamba model pretrained on PubChem achieving a high diversity with respect to all three metrics, and especially generates the highest number of unique generic scaffolds with a QED and sulphur avoidance score above 0.6. Mamba trained with PubChem also generates significantly more unique compounds and scaffolds that meet the 0.6 threshold than the decoder and RNN. While this balance between exploration and exploitation is important, it should be noted that the balance can also be refined through adjustment of RL hyperparameters.

Impact of architecture choice

Turning to the impact of architecture on RL performance, it is evident that all three examined architectures are capable of adapting effectively to specific tasks through RL. Notably, while all architectures quickly learn explicit token-based tasks such as the sulphur avoidance task, their behaviors diverge on more complex objectives. The decoder-only Transformer generally demonstrates greater fluctuation in scores across training epochs as well as over RL runs, compared to the RNN and Mamba models, which is significantly clear for the scores seen for the similarity guided task seen in Figure 4.15. These fluctuations, in combination with fast adaptation of the decoder model across tasks (see for example, Figures 4.10 and B.17) implies that the decoder responds more rapidly to the reward signal. Despite fluctuations, the

decoder exhibits particular proficiency in the similarity-guided task, achieving higher similarity scores than all other models (see Figures 4.13, 4.12, 4.14, and 4.15). In contrast, for the task of target activity against DRD2, no substantial performance differences are detected between architectures in terms of the attained scores, see Figure 4.25. For the other target activity tasks presented in Appendix B.3, there are some differences between the architectures: For the task of target activity towards JNK3, and the scores seen in Figure B.10, Mamba's target activity score increases more slowly than for the other architectures during the first steps. But this changes during the second half of the RL run, during which Mamba achieves higher scores than the other two models. For the task of target activity towards GSK3β, the decoder learns to generate compounds with a high target activity score faster than other models, indicated by the steep slope for the decoder's target activity score in Figure B.17. Following this steep increase, the decoder's score then drops below corresponding scores of the RNN and Mamba, before finally increasing to the same levels as the other models.

Overall, these observations suggest that while architectural choice does influence aspects of training dynamics and task-specific proficiency, there are no pronounced differences that generalize across tasks.

Diversity among generated compounds, scaffolds, and generic scaffolds varies across tasks. In the similarity-guided task, all models produce many unique compounds (Figure 4.16), but only a minuscule fraction surpass the similarity and QED score thresholds. Notably, the decoder and RNN trained with PubChem and randomized SMILES generate significantly more unique scaffolds, though most share the same generic structure, differing only in substituent groups (Figure 4.17 and 4.18). In contrast, using PubChem with canonical SMILES with these models results in scaffolds that are distinct even at the generic level.

The choice of training data

While the choice of architecture has a limited effect on the results of the RL, the influence of the training data is much more pronounced. When comparing ChEMBL and PubChem as pretraining sources, distinct differences emerge in learning efficiency and generative capacity. Models pretrained on ChEMBL consistently achieve higher scores more rapidly across architectures and tasks, see for example Figures 4.12, 4.13 and 4.14 for the similarity-guided task, suggesting that the more focused and medicinally relevant nature of the dataset facilitates more efficient exploration of the chemical space.

In contrast, models pretrained on the large PubChem dataset learn more slowly, likely due to the vast and complex chemical space that makes it harder to find high-scoring candidates (see Figures 4.22, 4.23, and 4.24). On the other hand, the models pretrained with PubChem have not converged fully as much as the ones trained with ChEMBL, which can be contributing to the slower learning of these models. Nevertheless, the advantages of PubChem are evident in the diversity of generated compounds. For example, the RNN and decoder pretrained with Pubchem for the similarity guided task, generate a higher number of unique compounds, scaffolds,

and generic scaffolds than corresponding models trained with PubChem, which can be seen in Figures 4.16, 4.17, and 4.18.

For the target activity task presented in the Results chapter (DRD2), PubChem-trained models generate slightly more unique compounds than ChEMBL-trained models (see Figure 4.26. Although, as the PubChem-trained models are slower in learning the task, fewer of the unique compounds reach a target activity and QED score above 0.6. Studying the number of unique scaffolds for the same task in Figure 4.27, similar patterns can be seen: PubChem-trained models generate more unique scaffolds, but a smaller fraction meets the 0.6 score threshold than for corresponding ChEMBL models. For the generic scaffolds generated under the target activity task towards DRD2 seen in Figure 4.28, there is no clear benefit from training with PubChem. Instead, the decoder and RNN trained with ChEMBL and canonical SMILES generate a comparable number of unique generic scaffolds to corresponding models trained with PubChem.

But for Mamba, the positive effects of training with PubChem are evident in the generic scaffolds as well: PubChem-trained Mamba models generate more unique generic scaffolds than corresponding ChEMBL-trained models. Interestingly, it seems that the Mamba models see a pronounced increase in generic scaffolds (see Figures 4.18 and 4.28) when training with PubChem, while on the other hand, the Mamba architecture sees a larger decrease in task-specific score when training with PubChem than other models (compare for example DRD2 scores for Mamba in Figure 4.24 with corresponding scores for decoder in Figure 4.23).

In summary, the choice of training data involves a trade-off between optimization efficiency and diversity yield. In general, ChEMBL enables models to learn more quickly and access high-scoring regions with greater ease, while PubChem expands the scope of generative diversity, albeit at the expense of learning speed and navigational complexity. The positive effects on diversity do, however, vary across architectures and optimization tasks. Throughout, these findings underscore the importance of aligning training data selection with specific project goals, whether the priority is rapid optimization or broad structural exploration.

Randomized SMILES

The incorporation of randomized SMILES during pretraining had varying effects depending on model architecture. For both the decoder-only Transformer and the RNN, randomized SMILES did not lead to substantial changes in optimization scores across tasks, for example, the DRD2 activity task seen in Figures 4.22 and 4.23, indicating that these architectures already have a strong capacity to abstract underlying chemical structures. In contrast, the Mamba architecture exhibited a pronounced benefit from training with randomized SMILES, see for example Figure 4.24, achieving significantly higher task-specific scores when being pretrained with randomized SMILES. This suggests that Mambas modeling approach may be more prone to memorize the training data and thus gain from the randomization over canonical SMILES. This aligns with the steep decrease in novelty seen when pretaining Mamba with canonical SMILES (see Figure 4.5e for the increase in novelty when training

with randomized SMILES).

Pretraining with randomized SMILES impacts molecular diversity in diverse ways across tasks, model architectures, and datasets. Considering the number of unique compounds generated, as illustrated in Figures 4.16 and 4.26, models trained with randomized SMILES consistently produce more unique compounds than their counterparts trained with canonical SMILES. Although the differences in the total number of unique compounds for both the similarity-guided and DRD2 tasks are relatively modest, the effect is more pronounced for the Mamba architecture on the DRD2 task: Mamba models pretrained with randomized SMILES yield significantly more unique compounds that meet the 0.6 score threshold than the Mamba models pretrained with canonical SMILES, as evident in Figure 4.26. For this task, this positive effect for Mamba extends to the number of unique scaffolds, as shown in Figure 4.27.

By contrast, for the similarity-guided task, randomized SMILES do not exert the same influence on scaffold and generic scaffold diversity. In fact, models pretrained on ChEMBL with canonical SMILES display higher diversity in unique scaffolds compared to those trained with randomized SMILES, as indicated by Figure 4.17. However, for PubChem-trained models in the similarity-guided task, randomization continues to positively impact the number of unique scaffolds (and quite dramatically so for the decoder and RNN). The trend for generic scaffolds in the similarity-guided task, shown in Figure 4.18, is distinctly inverted: both RNN and decoder models trained with PubChem generate more unique generic scaffolds when trained with canonical SMILES rather than randomized SMILES.

These findings highlight that the influence of randomized SMILES on diversity is highly context-dependent. For certain architectures and tasks, such as the Mamba model on the DRD2 task, randomized SMILES evidently promote the exploration of new compounds and scaffolds. The positive impact on diversity is likely due to the networks exposure to alternative representations, which encourages exploration of chemical space and mitigates memorization of specific token sequences. However, the effect is not universal and varies depending on the combination of dataset, architecture, and molecular diversity metric being considered. This suggests that the benefit of SMILES randomization is not uniform across all scenarios and should be strategically employed, with attention to the specific goals and context of the generative application.

Summary and Implications

In summary, the results of the RL experiments indicate that an effective pretraining strategy for generating high-scoring compounds is achieved by using the ChEMBL dataset with randomized SMILES, across all architectures and optimization tasks. In contrast, pretraining with the larger PubChem dataset serves as the optimal choice when compound diversity is the primary objective; however, this does not necessarily translate to scaffold diversity across all models, as discussed previously.

In addition to the models discussed thus far, the Decoder Mini, comprising 2.2 M parameters, was trained on the same optimization tasks as larger models. Across all

tasks, the Decoder Mini demonstrated competitive performance relative to the larger models with respect to both the desired properties, i.e., the scores, and molecular diversity. Results for the optimization task targeting DRD2 activity are presented in the Results section, while outcomes for the remaining tasks are provided in Appendix B.4, along with a brief discussion of these findings.

An important consideration in RL for molecular optimization is sample efficiency, which becomes especially crucial when employing more computationally expensive scoring functions than those evaluated in this thesis. If a generative model requires thousands of samples to learn a new task, direct optimization against these advanced scoring functions quickly becomes infeasible. The need for sample efficiency speaks in favor of the decoder architecture and pretraining with the smaller, more curated ChEMBL dataset.

It is important to recognize the limitations inherent in the oracle models utilized as scoring functions for target activity tasks (the DRD2, JNK3, and GSK3β described in Section 3.4). Oracle models are trained predictors that may be susceptible to extrapolation errors and other internal heuristics that do not fully correspond to experimental reality. This consideration is particularly relevant for applications of reinforcement learning in molecular design: while reinforcement learning offers powerful means for navigating chemical space and proposing novel candidates, the reliability of generated compounds ultimately depends on the accuracy, domain coverage, and calibration of the oracle models. In the broader context, the capacity of chemical language models and reinforcement learning to contribute to drug discovery is closely tied to the selection of training data as well as the robustness of property predictors.

For a less detailed summary of the discussion, see Appendix C.

6

Conclusion

In this thesis, we have systematically investigated how architectural choice, training data size and origin, and the use of randomized SMILES affect the behavior and performance of models for chemical language modeling. We explicitly compared three architectures: an RNN with LSTM cells, a decoder-only Transformer, and the Mamba architecture, utilizing two datasets: ChEMBL and PubChem. While the results are far from coherent, some overarching conclusions can be drawn. Overall, architectural choice had a limited effect on the generative performance, especially when provided a sufficient amount of data (training with PubChem). This suggests that the task of chemical language modeling might be a "too simple" of a task to motivate the use of more complex and computationally demanding architectures.

Training data proved to be the most influential factor shaping model performance. With ChEMBL and PubChem differing primarily in size and medicinal relevance (QED), ChEMBL-trained models more efficiently accessed high-scoring regions during RL, while PubChem-trained models achieved higher validity and uniqueness after pretraining, as well as broader compound diversity under RL. Evidently, PubChem provides models with a broader chemical space at the cost of making it more difficult to navigate.

Randomized SMILES consistently improved novelty across architectures and training data, refraining the models from memorizing the canonical syntax and instead generalizing better. The benefit of randomization was most pronounced for the Mamba architecture, both with respect to novelty and RL optimization. In addition, randomized SMILES generally improved compound diversity during RL, further speaking in favor of the use of randomized SMILES in chemical language modeling.

Reinforcement learning experiments revealed that all architectures can be utilized in RL tasks, but the sample efficiency and compound diversity were mainly governed by pretraining choices.

While the RNN and Mamba remain competitive, our thesis shows that under the training conditions studied, the decoder-only Transformer is generally preferable, as it achieves a higher validity and uniqueness, and presents higher sample efficiency during RL optimization.

Future Aspects

While this thesis offers insights into the effectiveness of different architectures and the influence of dataset size and source on model performance, the findings are far from conclusive. Many important aspects remain unexplored, and further research will be needed to fully understand the complexities of chemical language modeling and its practical applications. Below, a selection of what such further research could look like has been listed.

- **Hyperparameter Tuning:** Future studies should conduct thorough, systematic hyperparameter optimization for each architecture and dataset. Fine-tuning parameters such as learning rate, batch size, and parameters of the optimization algorithm may bring performance improvements and help address issues such as overfitting or slow convergence observed in this thesis.
- **Model Scaling:** A systematic investigation into optimal model scaling for different dataset sizes. Such an analysis could minimize training time while maintaining top performance across all metrics. Mitigating overfitting in models such as Mamba or RNN could further reduce loss during training and potentially improve performance in validity, uniqueness, and novelty, as well as in reinforcement learning tasks.
- **Training Data Curation:** The pronounced variation in model behavior and performance across different datasets suggests that further analyses would be beneficial. Datasets could be manually curated to balance diversity and medicinal relevance, thereby leveraging the relevance of ChEMBL with the size advantages of PubChem. Incorporating multiple randomized SMILES representations may further enhance the models' ability to generalize chemical syntax. This could also be extended into analyzing alternative molecular representations, such as graphs, beyond SMILES.
- **Multi-objective optimization:** Future research should explore the optimization of generative models using more complex oracle models as well as multi-objective reinforcement learning frameworks, incorporating additional properties beyond QED, to better reflect the complex trade-offs encountered in real-world drug discovery.

Researching these topics, we believe, would further enhance the generative models for the purpose of drug design. Ultimately, this thesis underscores the importance of data selection and augmentation in AI-driven molecular design, and we recognize that many opportunities remain for future progress in this field.

Bibliography

[1] JP Hughes, S Rees, SB Kalindjian, and KL Philpott. "Principles of early drug discovery". In: *British Journal of Pharmacology* 162.6 (2011), pp. 1239–1249. DOI: https://doi.org/10.1111/j.1476-5381.2010.01127.x.

[2] P G Polishchuk, T I Madzhidov, and A Varnek. "Estimation of the size of drug-like chemical space based on GDB-17 data." In: *Journal of computer-aided molecular design* 27.8 (2013), pp. 675–679. DOI: 10.1021/ci00057a005. URL: https://doi.org/10.1007/s10822-013-9672-4.

[3] Gisbert Schneider. "Automating drug discovery". In: *Nature Reviews Drug Discovery* 17.2 (2018). DOI: https://doi.org/10.1038/nrd.2017.232.

[4] Petra Schneider, W. Patrick Walters, Alleyn T. Plowright, Norman Sieroka, Jennifer Listgarten, Robert A. Goodnow, Jasmin Fisher, Johanna M. Jansen, José S. Duca, Thomas S. Rush, Matthias Zentgraf, John Edward Hill, Elizabeth Krutoholow, Matthias Kohler, Jeff Blaney, Kimito Funatsu, Chris Luebkemann, and Gisbert Schneider. "Rethinking drug design in the artificial intelligence era". In: *Nature Reviews Drug Discovery* 19.5 (2020). DOI: https://doi.org/10.1038/s41573-019-0050-3.

[5] Marwin H. S. Segler, Thierry Kogej, Christian Tyrchan, and Mark P. Waller. "Generating Focused Molecule Libraries for Drug Discovery with Recurrent Neural Networks". In: *ACS Central Science* 4.1 (2018). PMID: 29392184, pp. 120–131. DOI: 10.1021/acscentsci.7b00512. eprint: https://doi.org/10.1021/acscentsci.7b00512. URL: https://doi.org/10.1021/acscentsci.7b00512.

[6] Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Carlos Outeiral, Pedro Luis Cunha Farias, and Alán Aspuru-Guzik. *Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence Generation Models*. 2018. arXiv: 1705.10843 [stat.ML]. URL: https://arxiv.org/abs/1705.10843.

[7] Rocío Mercado, Tobias Rastemo, Edvard Lindelöf, Günter Klambauer, Ola Engkvist, Hongming Chen, and Esben Jannik Bjerrum. "Graph networks for molecular design". In: *Machine Learning: Science and Technology* 2.2 (Mar. 2021), p. 025023. DOI: 10.1088/2632-2153/abcf91. URL: https://dx.doi.org/10.1088/2632-2153/abcf91.

[8] Hongming Chen, Ola Engkvist, Yinhai Wang, Marcus Olivecrona, and Thomas Blaschke. "The rise of deep learning in drug discovery". In: *Drug Discovery Today* 23.6 (2018), pp. 1241–1250. ISSN: 1359-6446. DOI: https://doi.org/10.1016/j.drudis.2018.01.039. URL: https://www.sciencedirect.com/science/article/pii/S1359644617303598.

[9] Lewis Mervin, Samuel Genheden, and Ola Engkvist. "AI for drug design: From explicit rules to deep learning". In: *Artificial Intelligence in the Life Sciences* 2 (2022), p. 100041. ISSN: 2667-3185. DOI: https://doi.org/10.1016/j.ailsci.2022.100041. URL: https://www.sciencedirect.com/science/article/pii/S2667318522000113.

[10] Xiangxiang Zeng, Fei Wang, Yuan Luo, Seung-gu Kang, Jian Tang, Felice C. Lightstone, Evandro F. Fang, Wendy Cornell, Ruth Nussinov, and Feixiong Cheng. "Deep generative molecular design reshapes drug discovery". In: *Cell Reports Medicine* 3.12 (2022), p. 100794. ISSN: 2666-3791. DOI: https://doi.org/10.1016/j.xcrm.2022.100794. URL: https://www.sciencedirect.com/science/article/pii/S2666379122003494.

[11] Joshua Meyers, Benedek Fabian, and Nathan Brown. "De novo molecular design and generative models". In: *Drug Discovery Today* 26.11 (2021), pp. 2707–2715. ISSN: 1359-6446. DOI: https://doi.org/10.1016/j.drudis.2021.05.019. URL: https://www.sciencedirect.com/science/article/pii/S1359644621002531.

[12] Viraj Bagal, Rishal Aggarwal, P. K. Vinod, and U. Deva Priyakumar. "MolGPT: Molecular Generation Using a Transformer-Decoder Model". In: *Journal of Chemical Information and Modeling* 62.9 (2022). PMID: 34694798, pp. 2064–2076. DOI: 10.1021/acs.jcim.1c00600. eprint: https://doi.org/10.1021/acs.jcim.1c00600. URL: https://doi.org/10.1021/acs.jcim.1c00600.

[13] Varnavas D. Mouchlis, Antreas Afantitis, Angela Serra, Michele Fratello, Anastasios G. Papadiamantis, Vassilis Aidinis, Iseult Lynch, Dario Greco, and Georgia Melagraki. "Advances in de Novo Drug Design: From Conventional to Machine Learning Methods". In: *International Journal of Molecular Sciences* 22.4 (2021), p. 1676. DOI: 10.3390/ijms22041676.

[14] Hannes H Loeffler, Jiazhen He, Alessandro Tibo, Jon Paul Janet, Alexey Voronov, Lewis H. Mervin, and Ola Engkvist. "Reinvent 4: Modern AIdriven generative molecule design". In: *Journal of Cheminformatics* 16.1 (2024). URL: https://doi.org/10.1186/s13321-024-00812-5.

[15] Caterina Bissantz, Bernd Kuhn, and Martin Stahl. "A Medicinal Chemists Guide to Molecular Interactions". In: *Journal of Medicinal Chemistry* 53.14 (2010). PMID: 20345171, pp. 5061–5084. DOI: 10.1021/jm100112j. URL: https://doi.org/10.1021/jm100112j.

[16] EMBL-EBIChEMBL. *ChEMBL35*. data retrieved from EMBL-EBI, http://www.ebi.ac.uk/chembl. 2024.

[17] NIH. *Pubchem*. https://pubchem.ncbi.nlm.nih.gov/. 2025.

[18] Laurianne David, Amol Thakkar, Rocío Mercado, and Ola Engkvist. "Molecular representations in AI-driven drug discovery: a review and practical guide". In: *Journal of Cheminformatics* 12.1 (2020). DOI: https://doi.org/10.1186/s13321-020-00460-5.

[19] David Weininger. "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules". In: *Journal of Chemical Information and Computer Sciences* 28.1 (1988), pp. 31–36. ISSN: 0095-2338. DOI: 10.1021/ci00057a005. URL: https://doi.org/10.1021/ci00057a005.

[20] Alex Sherstinsky. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network". In: *Physica D: Nonlinear Phenomena* 404.132306 (2020). URL: https://www.sciencedirect.com/science/article/pii/S0167278919305974.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[22] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. "Improving Language Understanding by Generative Pre-Training". In: (2018). OpenAI. URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[23] Albert Gu and Tri Dao. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. 2024. arXiv: 2312.00752 [cs.LG]. URL: https://arxiv.org/abs/2312.00752.

[24] Guy W. Bemis and Mark A. Murcko. "The Properties of Known Drugs. 1. Molecular Frameworks". In: *Journal of Medicinal Chemistry* 39.15 (1996). PMID: 8709122, pp. 2887–2893. DOI: 10.1021/jm9602928. eprint: https://doi.org/10.1021/jm9602928. URL: https://doi.org/10.1021/jm9602928.

[25] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[26] Gemini Team et al. *Gemini: A Family of Highly Capable Multimodal Models*. 2024. arXiv: 2312.11805 [cs.CL]. URL: https://arxiv.org/abs/2312.11805.

[27] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL]. URL: https://arxiv.org/abs/2307.09288.

[28] OpenAI. *Finetune Transformer LM*. https://github.com/openai/finetune-transformer-lm. 2018.

[29] OpenAI. *GPT-2*. https://github.com/openai/gpt-2. 2019.

[30] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry,

Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL: https://arxiv.org/abs/2005.14165.

[31] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774.

[32] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL]. URL: https://arxiv.org/abs/1409.0473.

[33] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. *On Layer Normalization in the Transformer Architecture*. 2020. arXiv: 2002.04745 [cs.LG]. URL: https://arxiv.org/abs/2002.04745.

[34] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. *RoFormer: Enhanced Transformer with Rotary Position Embedding*. 2023. arXiv: 2104.09864 [cs.CL]. URL: https://arxiv.org/abs/2104.09864.

[35] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Re. *HiPPO: Recurrent Memory with Optimal Polynomial Projections*. 2020. arXiv: 2008.07669 [cs.LG]. URL: https://arxiv.org/abs/2008.07669.

[36] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. *Combining Recurrent, Convolutional, and Continuous-time Models with Linear State-Space Layers*. 2021. arXiv: 2110.13985 [cs.LG]. URL: https://arxiv.org/abs/2110.13985.

[37] Albert Gu, Karan Goel, and Christopher Ré. *Efficiently Modeling Long Sequences with Structured State Spaces*. 2022. arXiv: 2111.00396 [cs.LG]. URL: https://arxiv.org/abs/2111.00396.

[38] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv preprint arXiv:1412.6980* (2014). Published as a conference paper at ICLR 2015. URL: https://arxiv.org/abs/1412.6980.

[39] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018.

[40] Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. "Molecular de-novo design through deep reinforcement learning". In: *Journal of Cheminformatics* 9.1 (2017). URL: https://doi.org/10.1186/s13321-017-0235-x.

[41] Hampus Gummesson Svensson, Christian Tyrchan, Ola Engkvist, and Morteza Haghir Chehreghani. *Diversity-Aware Reinforcement Learning for de novo Drug Design*. 2024. arXiv: 2410.10431 [cs.LG]. URL: https://arxiv.org/abs/2410.10431.

[42] Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. "Molecular de-novo design through deep reinforcement learning". In: *Journal of Cheminformatics* 9.1 (2017), p. 48. ISSN: 1758-2946. DOI: 10.1186/s13321-017-0235-x. URL: https://doi.org/10.1186/s13321-017-0235-x.

[43] J. T. Cohen, D. C. Bellinger, and B. A. Shaywitz. "A quantitative analysis of prenatal methyl mercury exposure and cognitive development". In: *American Journal of Preventive Medicine* 29.4 (Nov. 2005), pp. 353–365. DOI: 10.1016/j.amepre.2005.06.007.

[44] Miklos Feher and Jonathan M. Schmidt. "Property Distributions: Differences between Drugs, Natural Products, and Molecules from Combinatorial Chemistry". In: *Journal of Chemical Information and Computer Sciences* 43.1 (2003). PMID: 12546556, pp. 218–227. DOI: 10.1021/ci0200467. eprint: https://doi.org/10.1021/ci0200467. URL: https://doi.org/10.1021/ci0200467.

[45] Michelle W. Y. Southey and Michael Brunavs. "Introduction to small molecule drug discovery and preclinical development". In: *Frontiers in Drug Discovery* Volume 3 - 2023 (2023). ISSN: 2674-0338. DOI: 10.3389/fddsv.2023.1314077. URL: https://www.frontiersin.org/journals/drug-discovery/articles/10.3389/fddsv.2023.1314077.

[46] Richard D. Taylor, Malcolm MacCoss, and Alastair D. G. Lawson. "Rings in Drugs". In: *Journal of Medicinal Chemistry* 57.14 (2014). Published 2014/07/24, pp. 5845–5859. ISSN: 0022-2623. DOI: 10.1021/jm4017625. URL: https://doi.org/10.1021/jm4017625.

[47] MolecularAI. *REINVENT4*. https://github.com/MolecularAI/REINVENT4. GitHub repository, accessed: 2024-04-07. 2022.

[48] Adam Paszke, Sam Gross, Francesco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f701272774Paper.pdf.

[49] Jeff Guo and Philippe Schwaller. *Saturn: Sample-efficient Generative Molecular Design using Memory Manipulation*. 2024. arXiv: 2405.17066 [q-bio.BM]. URL: https://arxiv.org/abs/2405.17066.

[50] Phil Wang. *rotary-embedding-torch*. https://github.com/lucidrains/rotary-embedding-torch. GitHub repository, accessed: 2024-05-16. 2021.

[51] Albert Gu, Karan Goel, Tri Dao, Atri Rudra, Christopher Ré, et al. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. https://github.com/state-spaces/mamba. Accessed: 2024-07-31. 2023.

[52] Brandon Cohen and Charles V. Preuss. *Celecoxib*. 2024. URL: https://www.ncbi.nlm.nih.gov/books/NBK535359/.

[53] David Rogers and Mathew Hahn. "Extended-Connectivity Fingerprints". In: *Journal of Chemical Information and Modeling* 50.5 (2010). URL: https://doi.org/10.1021/ci100050t.

[54] Jian-Ping Zhang, Todd Lencz, and Anil K. Malhotra. "D2 receptor genetic variation and clinical response to antipsychotic drug treatment: a meta-analysis".

In: *The American Journal of Psychiatry* 167.7 (2010), pp. 763–772. DOI: 10.1176/appi.ajp.2009.09040598.

[55] Yibo Li, Liangren Zhang, and Zhenming Liu. "Multi-objective de novo drug design with conditional graph generative model". In: *Journal of Cheminformatics* 10.1 (July 2018), p. 33. ISSN: 1758-2946. DOI: 10.1186/s13321-018-0287-6. URL: https://doi.org/10.1186/s13321-018-0287-6.

[56] G. R. Bickerton, G. V. Paolini, J. Besnard, S. Muresan, and A. L. Hopkins. "Quantifying the chemical beauty of drugs". In: *Nature Chemistry* 4.2 (2012), pp. 90–98. DOI: 10.1038/nchem.1243.

[57] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. *Understanding deep learning requires rethinking generalization*. 2017. arXiv: 1611.03530 [cs.LG]. URL: https://arxiv.org/abs/1611.03530.

[58] Josep Arús-Pous, Simon Viet Johansson, Oleksii Prykhodko, Esben Jannik Bjerrum, Christian Tyrchan, Jean-Louis Reymond, Hongming Chen, and Ola Engkvist. "Randomized SMILES strings improve the quality of molecular generative models". In: *Journal of Cheminformatics* 11.1 (2019), p. 71. DOI: 10.1186/s13321-019-0393-0. URL: https://doi.org/10.1186/s13321-019-0393-0.

A

Additional methods

A.1 Transformations in Preprocessing Pipeline

Table A.1: Transformations in the filtering pipeline. These transformations help standardize molecules when RDKit is not sufficient.

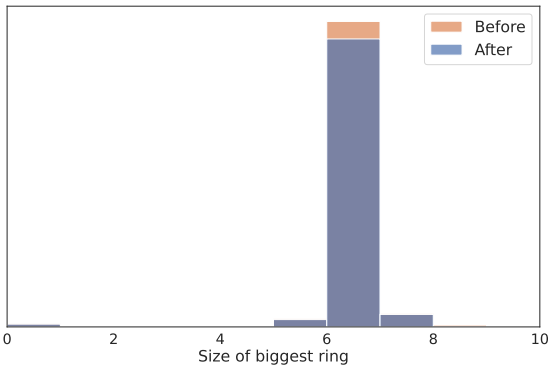
Transformation	**Description**
Nitro to N+(O-)=O	[N,P,As,Sb;X3:1](=[O,S,Se,Te:2])=[O,S,Se,Te:3] »[*+1:1]([*-1:2])=[*:3]
Sulfone to S(=O)(=O)	[S+2:1]([O-:2])([O-:3]) »[S+0:1](=[O-0:2])(=[O-0:3])
Pyridine oxide to n+O-	[nH0+0:1]=[OH0+0:2] »[n+:1][O-:2]
Azide to N=N+=N-	[*:1][N:2]=[N:3]#[N:4] »[*:1][N:2]=[N+:3]=[N-:4]
Diazo/azo to =N+=N-	[*:1]=[N:2]#[N:3] »[*:1]=[N+:2]=[N-:3]
Sulfoxide to -S+(O-)-	[!O:1][S+0;X3:2](=[O:3])[!O:4] »[*:1][S+1:2]([O-:3])[*:4]
Phosphate to P(O-)=O	[O,S,Se,Te;-1:1][P+;D4:2][O,S,Se,Te;-1:3] »[*+0:1]=[P+0;D5:2][*-1:3]
C/S+N to C/S=N+	[C,S&!$([S+]-[O-]);X3+1:1]([NX3:2])[NX3!H0:3] »[*+0:1]([N:2])=[N+:3]
P+N to P=N+	[P;X4+1:1]([NX3:2])[NX3!H0:3] »[*+0:1]([N:2])=[N+:3]
Normalize hydrazine-diazonium	[CX4:1][NX3H:2]- [NX3H:3][CX4:4][NX2+:5]#[NX1:6] »[CX4:1][NH0:2]=[NH+:3][C:4][N+0:5]=[NH:6]
Recombine 1,3-separated charges	[N,P,As,Sb,O,S,Se,Te;-1:1]- [A+0:2]=[N,P,As,Sb,O,S,Se,Te;+1:3] »[*-0:1]=[*:2]-[*+0:3]
Recombine 1,3-separated charges	[n,o,p,s;-1:1]:[a:2]=[N,O,P,S;+1:3] »[*-0:1]:[*:2]-[*+0:3]
Recombine 1,3-separated charges	[N,O,P,S;-1:1]-[a:2]:[n,o,p,s;+1:3] »[*-0:1]=[*:2]:[*+0:3]

Recombine 1,5-separated charges	[N,P,As,Sb,O,S,Se,Te;-1:1]-[A+0:2]=[A:3]-[A:4]=[N,P,As,Sb,O,S,Se,Te;+1:5] »[*-0:1]=[*:2]-[*:3]=[*:4]-[*+0:5]
Recombine 1,5-separated charges	[n,o,p,s;-1:1]:[a:2]:[a:3]:[c:4]=[N,O,P,S;+1:5] »[*-0:1]:[*:2]:[*:3]:[c:4]-[*+0:5]
Recombine 1,5-separated charges	[N,O,P,S;-1:1]-[c:2]:[a:3]:[a:4]:[n,o,p,s;+1:5] »[*-0:1]=[c:2]:[*:3]:[*:4]:[*+0:5]
Normalize 1,3 conjugated cation	[N,O!$(*N);+0!H0:1]-[A:2]=[N!$(* [N,O,P,S;-1]),O;+1H0:3] »[*+1:1]=[*:2]-[*+0:3]
Normalize 1,3 conjugated cation	[n;+0!H0:1]:[c:2]=[N!$(* [N,O,P,S;-1]),O;+1H0:3] »[*+1:1]:[*:2]-[*+0:3]
Normalize 1,5 conjugated cation	[N,O!$(*N);+0!H0:1]-[A:2]=[A:3]-[A:4]=[N!$(* [N,O,P,S;-1]),O;+1H0:5] »[*+1:1]=[*:2]-[*:3]=[*:4]-[*+0:5]
Normalize 1,5 conjugated cation	[n;+0!H0:1]:[a:2]:[a:3]:[c:4]=[N!$(* [N,O,P,S;-1]),O;+1H0:5] »[n+1:1]:[*:2]:%[*:3]:[*:4]-[*+0:5]
Charge normalization	[F,Cl,Br,I,At;-1:1]=[O:2] »[*-0:1][O-:2]
Charge recombination	[N,P,As,Sb;-1:1]=[C+;v3:2]»[*+0:1]

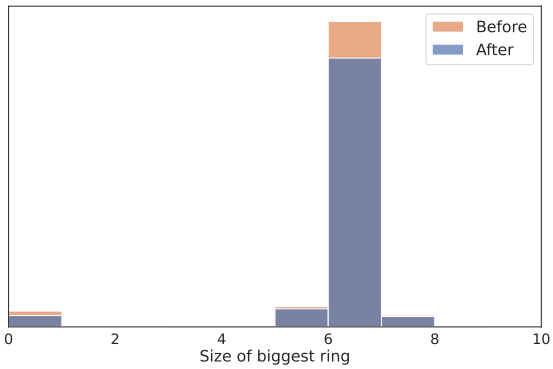
B

Additional Results

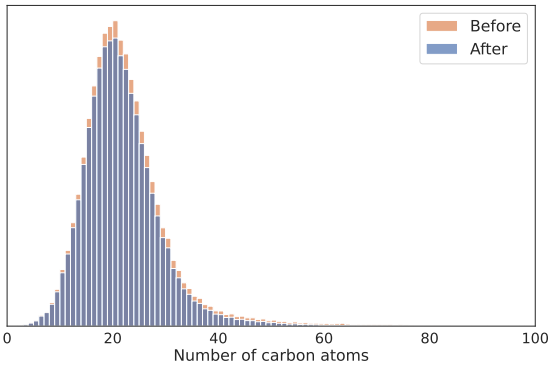
B.1 Additional Distributions of the Training Data



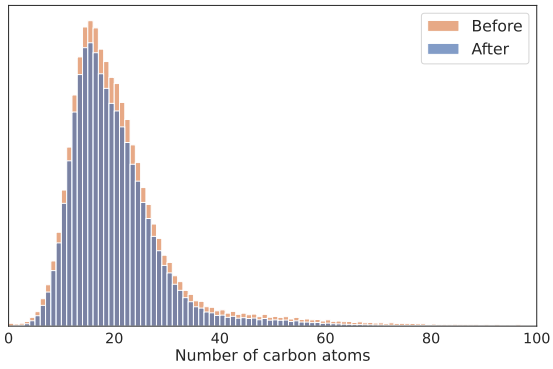
(a) Size of biggest ring (ChEMBL)



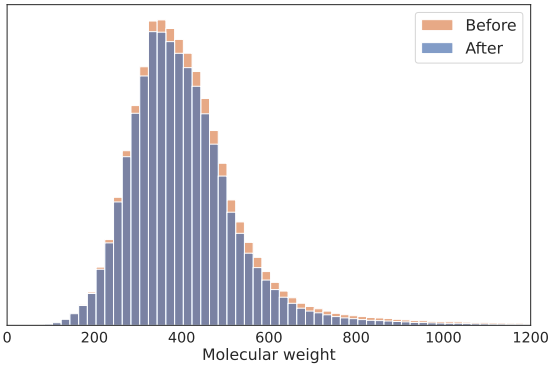
(b) Size of biggest ring (PubChem)



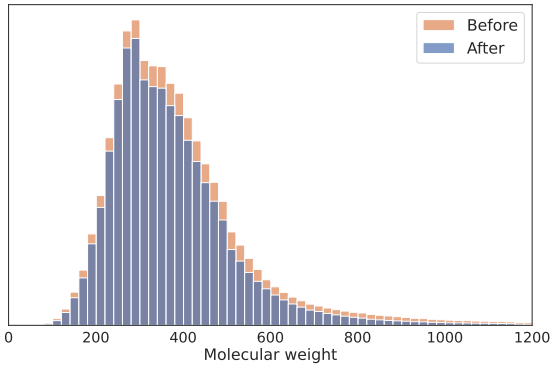
(c) Number of carbon atoms (ChEMBL)



(d) Number of carbon atoms (PubChem)



(e) Molecular weight (ChEMBL)

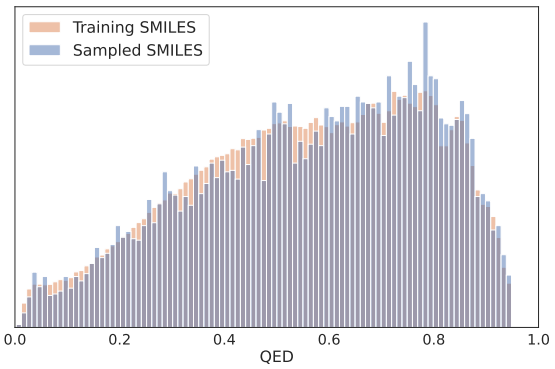


(f) Molecular weight (PubChem)

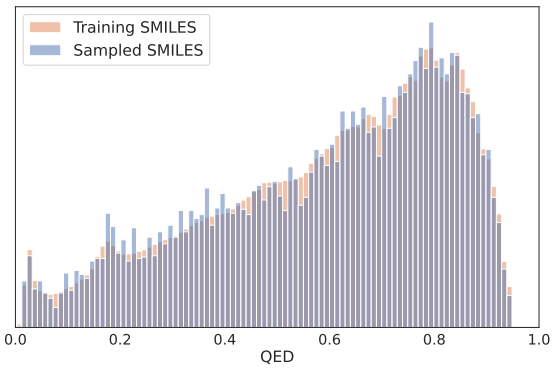
Figure B.1: Additional distributions of attributes of the training data before (orange) and after (blue) preprocessing.

B.2 The Generated Molecules and Their Attributes

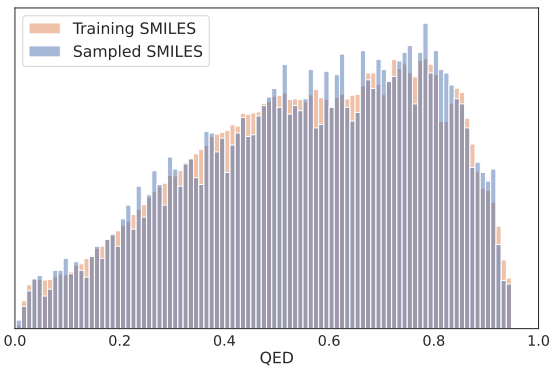
This section only concerns models trained with canonical SMILES. After pretraining, a set of 10.000 compounds was sampled from each model's lowest validation loss epoch, and a set of chemical properties was computed. In Figures B.2, B.3, and B.4, the distributions of these properties can be seen for the sampled compounds and the corresponding distribution within the training data. For all studied properties, all models reproduce distributions similar to the training data.



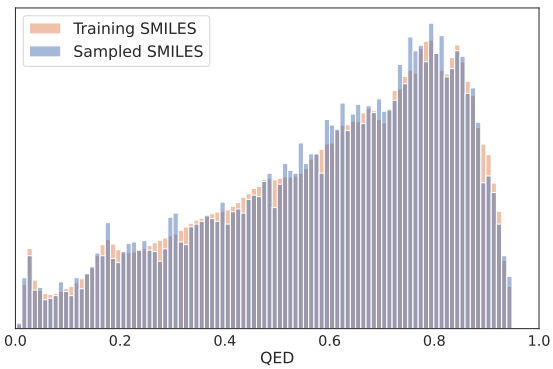
(a) Decoder (ChEMBL)



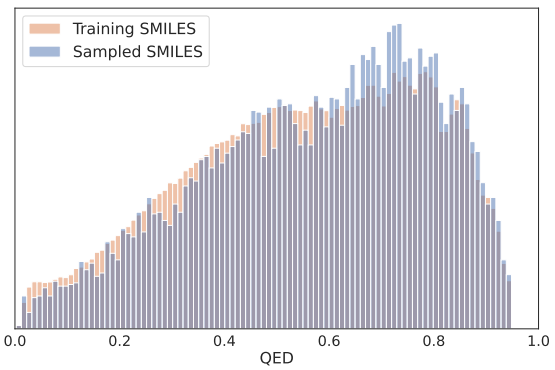
(b) Decoder (PubChem)



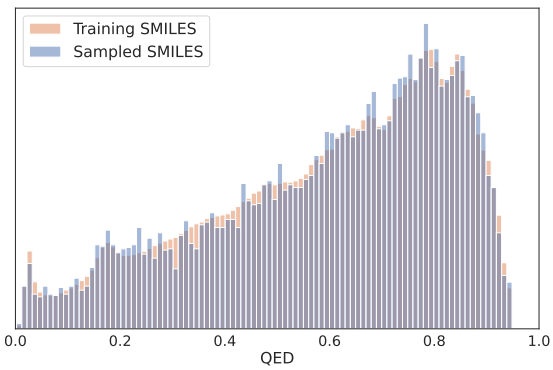
(c) RNN (ChEMBL)



(d) RNN (PubChem)

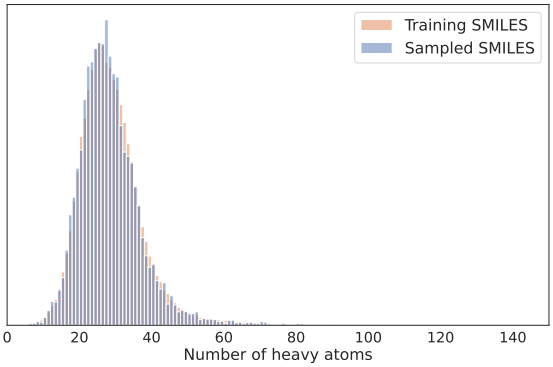


(e) Mamba (ChEMBL)

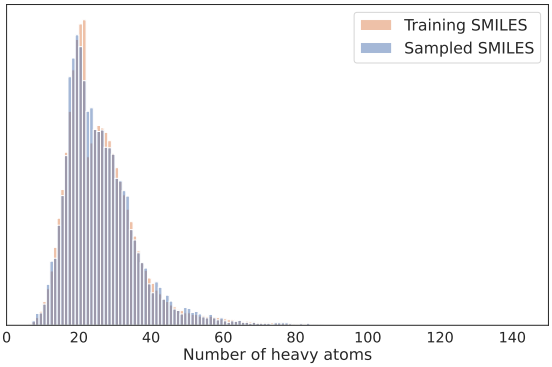


(f) Mamba (PubChem)

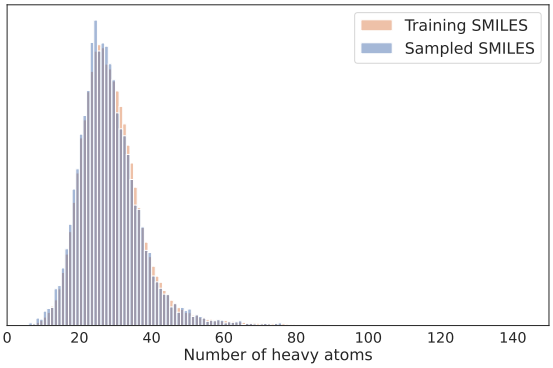
Figure B.2: QED distribution of sampled molecules and training data. All models reproduce the distribution of the training datasets similarly.



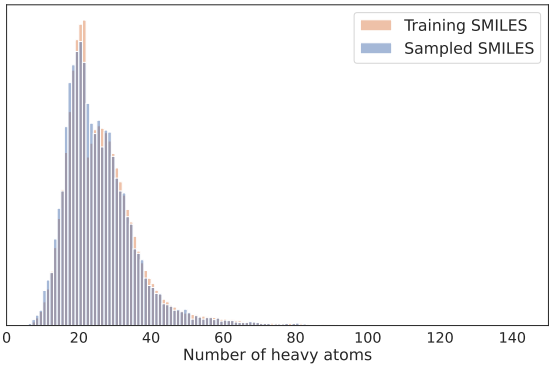
(a) Decoder (ChEMBL)



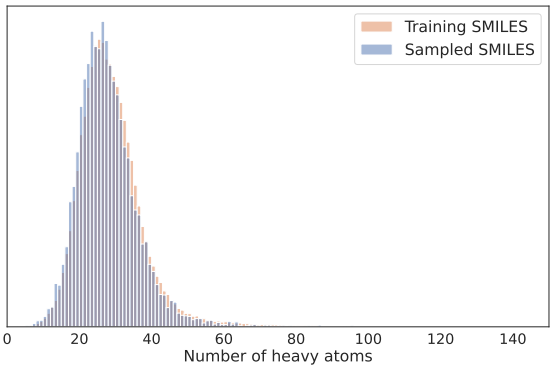
(b) Decoder (PubChem)



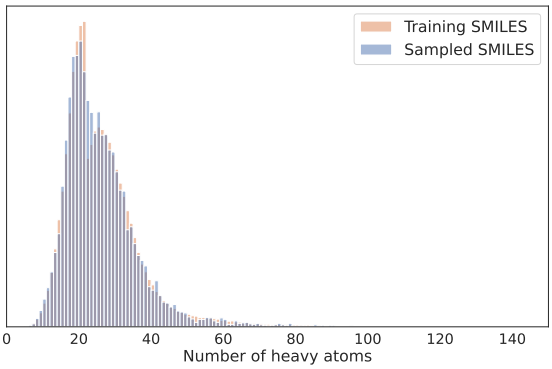
(c) RNN (ChEMBL)



(d) RNN (PubChem)

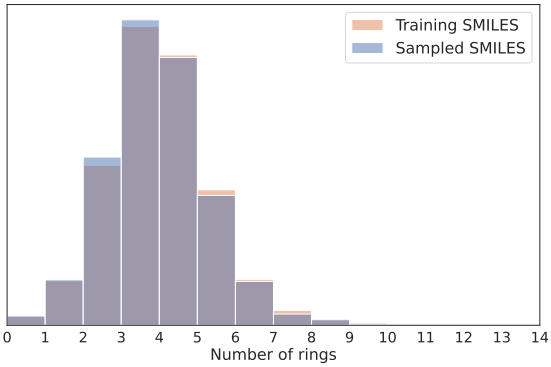


(e) Mamba (ChEMBL)

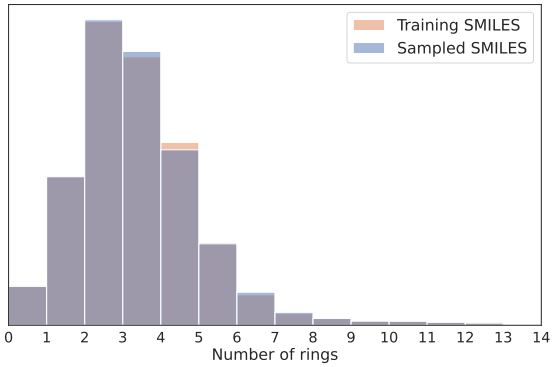


(f) Mamba (PubChem)

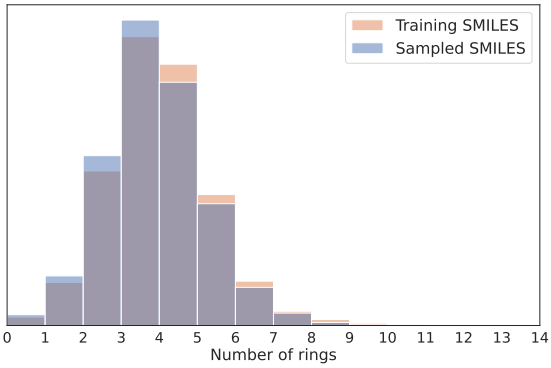
Figure B.3: Number of heavy atoms distribution of sampled molecules and training data. All models reproduce the distribution of the training datasets similarly.



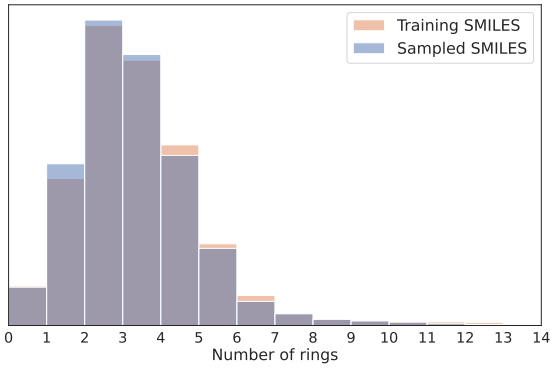
(a) Decoder (ChEMBL)



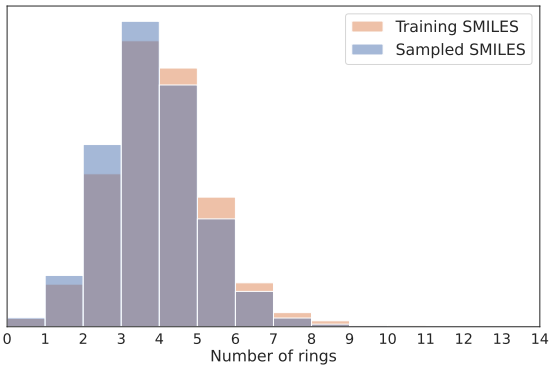
(b) Decoder (PubChem)



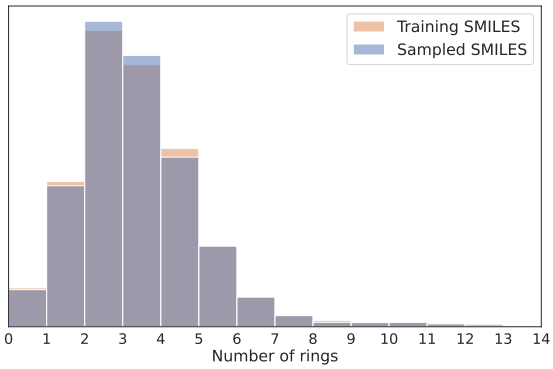
(c) RNN (ChEMBL)



(d) RNN (PubChem)



(e) Mamba (ChEMBL)



(f) Mamba (PubChem)

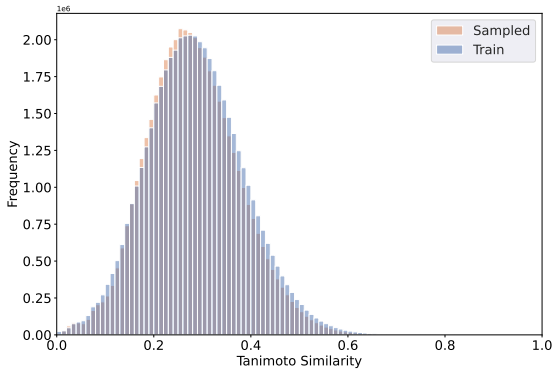
Figure B.4: Number of rings distribution of sampled molecules and training data. All models reproduce the distribution of the training datasets similarly.

In an attempt to quantify the similarity between the distributions, the KL-divergence between the distributions was computed. The resulting KL-divergences are listed in Table B.1. For all model-training data combinations, the KL-divergence is smaller than 0.03; however, there is no evident pattern as to which of the models better approximates the training distributions.

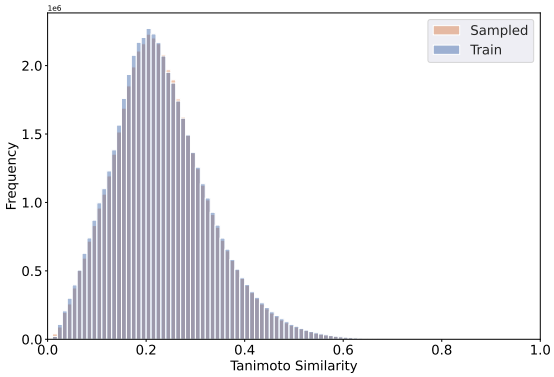
	QED	Molecular weight	Number of rings
Decoder - ChEMBL	0.0123	0.0299	0.0024
RNN - ChEMBL	0.0112	0.0271	0.0080
Mamba - ChEMBL	0.0160	0.03817	0.01557
Decoder - PubChem	**0.0091**	0.0106	**0.0011**
RNN - PubChem	0.0103	0.0087	0.0050
Mamba - PubChem	0.0097	**0.0075**	0.0024

Table B.1: KL-divergence between attributes among sampled molecules and the molecules in the training set. The distributions for each attribute were constructed by computing the corresponding attribute for each molecule in the set (sampled or training), and binning the results into bins of size 0.01 for QED, 20 for molecular weight, and 1 for number of rings. A Lower KL-divergence means that the distributions are more similar; the lowest KL-divergence for each attribute has been marked in bold.

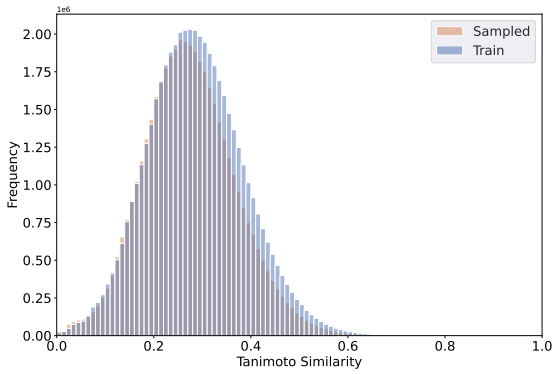
In addition to the set of molecular properties among sampled molecules, the pairwise similarities between molecules was computed as well. The pairwise similarities together with corresponding pairwise similarities between training samples can be seen in Figure B.5. Similar to the other properties, the distribution of pairwise similarities among sampled molecules resembles that of the training distribution.



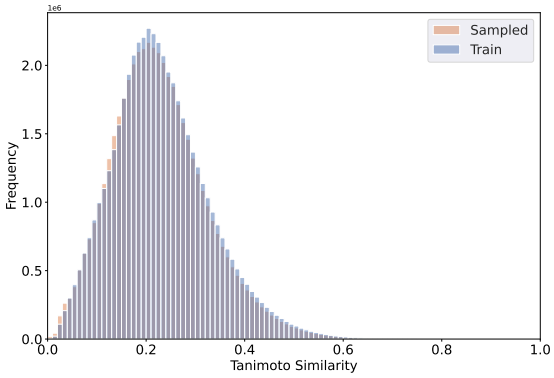
(a) Decoder (ChEMBL)



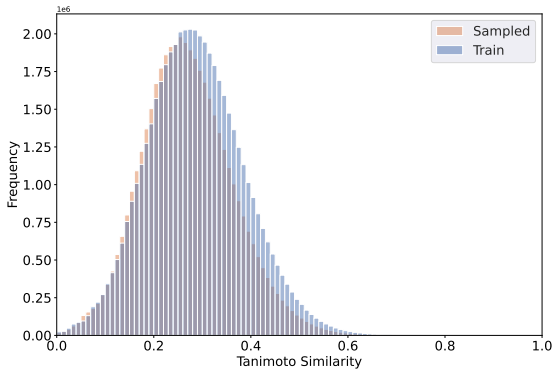
(b) Decoder (PubChem)



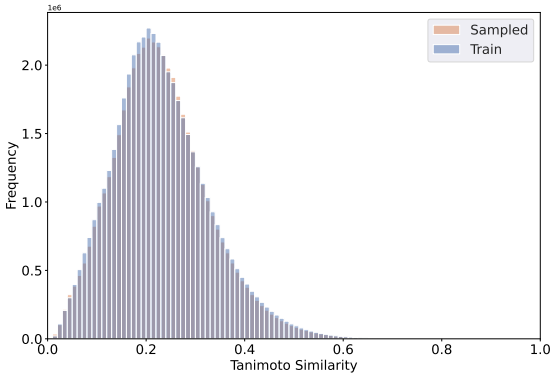
(c) RNN (ChEMBL)



(d) RNN (PubChem)



(e) Mamba (ChEMBL)



(f) Mamba (PubChem)

Figure B.5: Pairwise similarities between sampled molecules (orange) and training data (blue). The pairwise similarity was computed using Tanimoto similarity. For all four models, the distributions are similar for the sampled compounds and the training compounds; however, for the ChEMBL models, there is a small shift to the left for the sampled compounds compared to the distribution for the training compounds.

In Figure B.6, a subset of the molecules generated by the models can be seen.

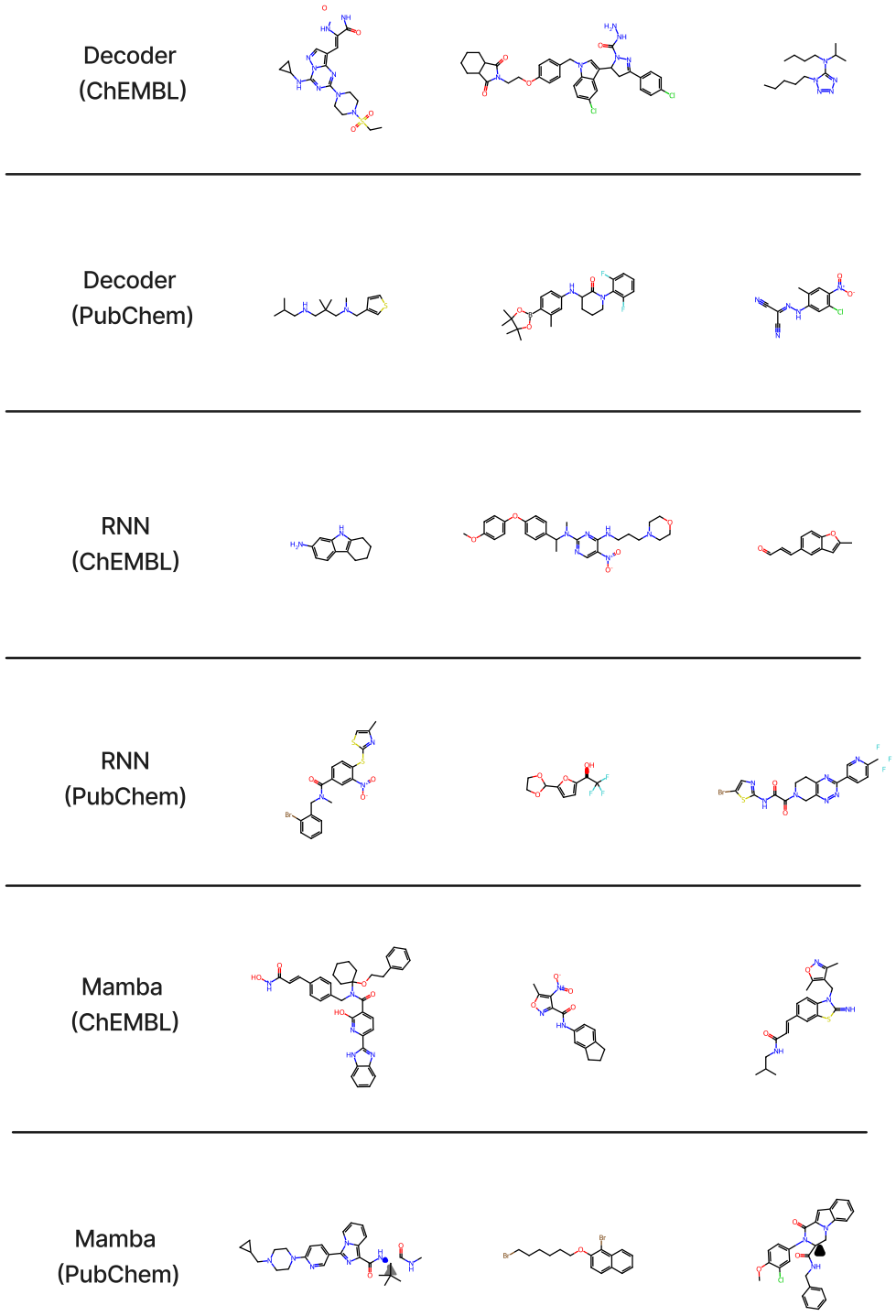


Figure B.6: Molecules sampled by the trained models. For each model, a set of 10^4 molecules was sampled, out of which the three seen for each model in this figure were uniformly sampled from this set.

B.3 Additional RL Tasks

In addition to the optimization tasks presented in the report, RL was run for two other target activity tasks, namely: JNK3 and GSK3β.

B.3.1 JNK3

To compare how well the models adapted to RL optimization towards activity against the mitogen-activated protein kinase JNK3, a random forest classifier was used as described in Section 3.4. Firstly, the impact from the dataset was evaluated on each of the models, and finally, an evaluation between the three models was made using the model from each architecture that was pretrained on ChEMBL using randomized SMILES. In Figure B.7, the average score and QED over steps in RL for target activity JNK3 for the RNN across datasets is shown.

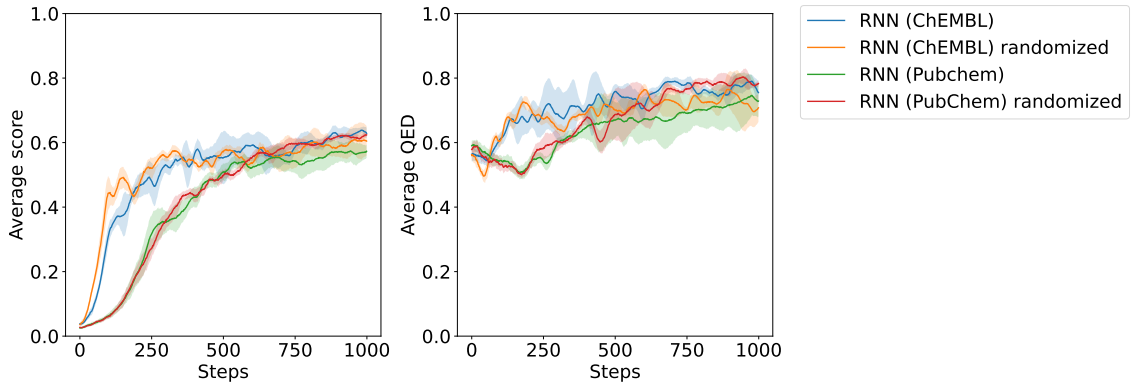


Figure B.7: RNN average score and QED over steps in RL for target activity (JNK3) across datasets. The models pretrained on ChEMBL adapt faster to the task as compared to the models pretrained on PubChem, as seen on the average score converging around 0.6. All models adapt similarly to the QED around 0.7.

The RNN pretrained on ChEMBL adapts relatively fast to the task, reaching average scores close to convergence after 250 steps. Those pretrained on PubChem reach those scores after around 600 steps.

In Figure B.8, the average score and QED over steps in RL for target activity JNK3 for the Decoder across datasets are shown.

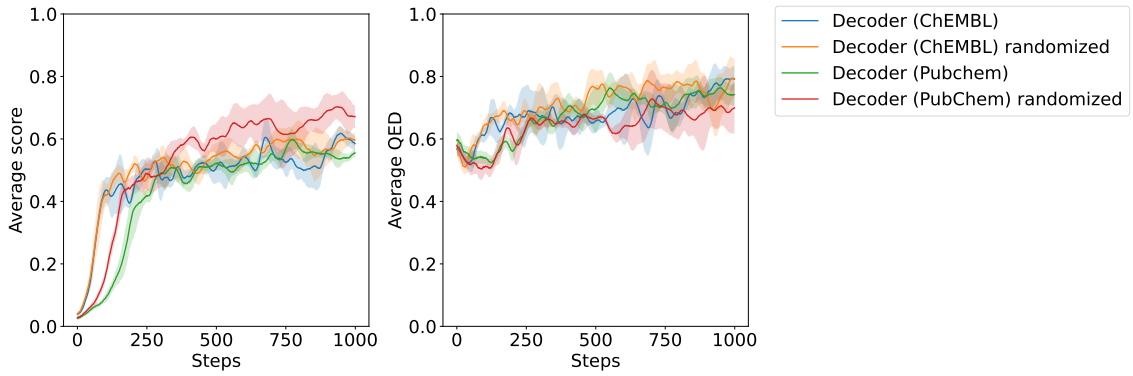


Figure B.8: Decoder average score and QED over steps in RL for target activity (JNK3) across datasets. The models pretrained on ChEMBL adapt faster to the task as compared to the models pretrained on PubChem, as seen in the average score converging around 0.6. The Decoder pretrained on PubChem with randomized SMILES does, however, increase in average score towards the final steps more than the other models. All models adapt similarly to the QED around 0.7.

Similarly to the RNN, the Decoders pretrained on ChEMBL learns the task faster, but this difference is not as prominent as for the RNN as per Figure B.7. The Deoder pretrained on PubChem with randomized SMILES increase in average score towards the final steps of the RL and show fewer signs of convergence than the others.

In Figure B.9, the average score and QED over steps in RL for target activity JNK3 for the Mamba across datasets are shown.

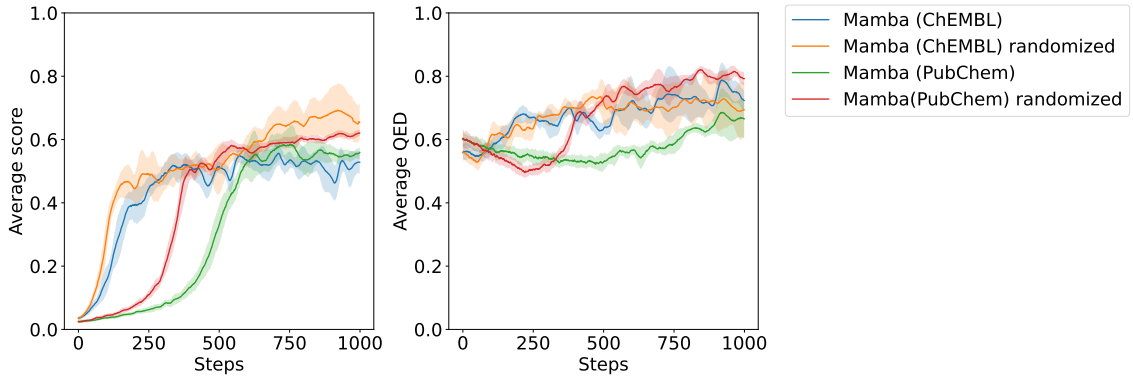


Figure B.9: Mamba average score and QED over steps in RL for target activity (JNK3) across datasets. The models pretrained on ChEMBL adapt faster to the task as compared to the models pretrained on PubChem, as seen on the average score converging around 0.6. All models adapt similarly to the QED around 0.7.

The Mamba models show greater variance in adapting to the task according to the average score compared to previous tasks. As for the other model architectures, the models pretrained on ChEMBL learns the task during less steps as those pretrained on PubChem. For Mamba, the model pretrained on ChEMBL with randomized SMILES has the highest average score at the final step of the RL.

In Figure B.10, the average score and QED over steps in RL for target activity JNK3 for the Decoder, RNN, and Mamba pretrained on ChEMBL randomized SMILES are shown.

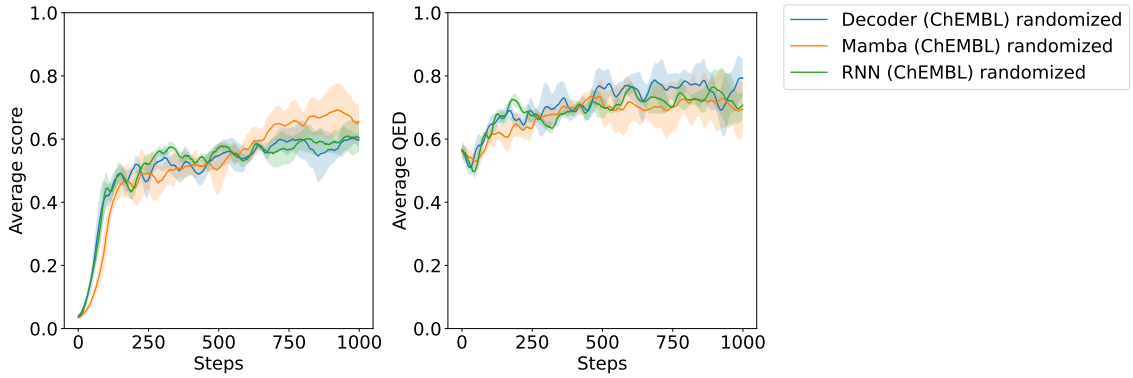


Figure B.10: Decoder, RNN, and Mamba average score and QED over steps in RL for target activity (JNK3) pretrained on ChEMBL randomized SMILES. The difference in average score and QED is minuscule across the model architectures, all converging at an average score of 0.6, reaching a QED of 0.7 at a similar pace.

When comparing the models pretrained on the same dataset (ChEMBL randomized SMILES), the difference in adapting to the task based on average score and QED is minuscule.

When evaluating the number of unique compounds over the 1000 steps per model-dataset pair, most models generate similar amounts of unique compounds, all generating around 60.000. In Figure B.11, this distribution can be seen. When including a threshold of a compound also having a score and QED > 0.6, few of their generated unique compounds meet this criterion, which can also be seen in the figures above, for example Figure B.10, since the average score for even the best performing model-dataset pair don't exceed 0.6 with a large margin.

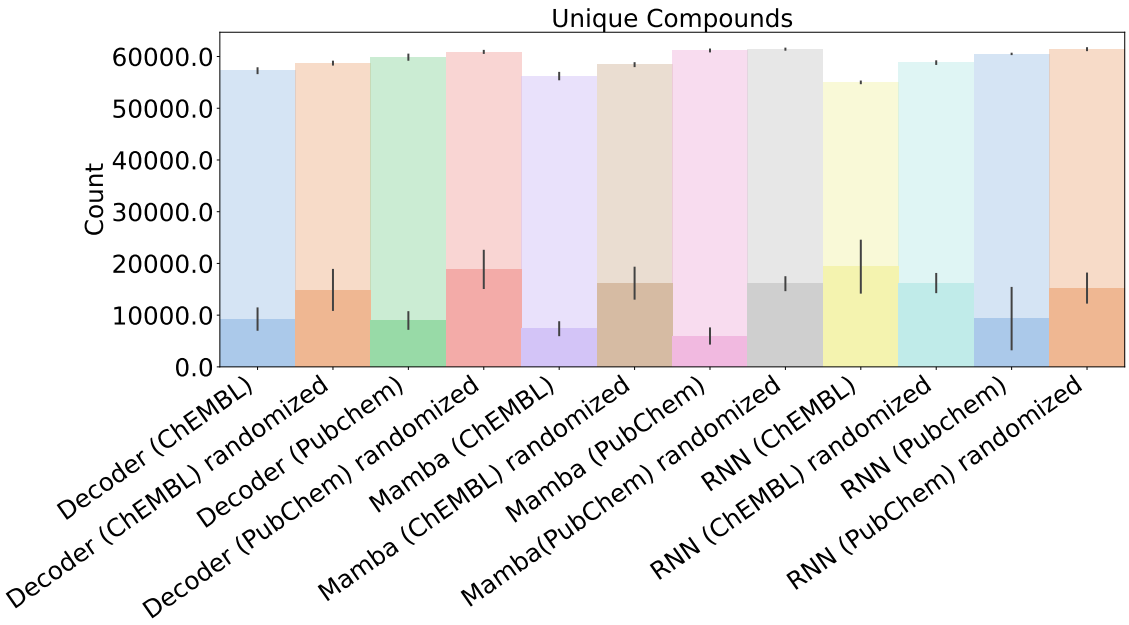


Figure B.11: Number of unique compounds generated by all the models pretrained on both datasets with and without randomized SMILES over 1000 steps with a batch size of 64 in RL for target activity (JNK3). The total number of compounds is illustrated by the transparent bar, and the solid bar represents the number of compounds that also have a score and QED over 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these three runs).

Figure B.12 shows the number of unique scaffolds over the 1000 steps per model-dataset pair. The amount of unique scaffolds varies between the models. PubChem with randomized SMILES helps the model generate the largest number of unique scaffolds. When including a threshold of a compound also having a score and QED > 0.6, the behavior varies between the model-dataset pairs.

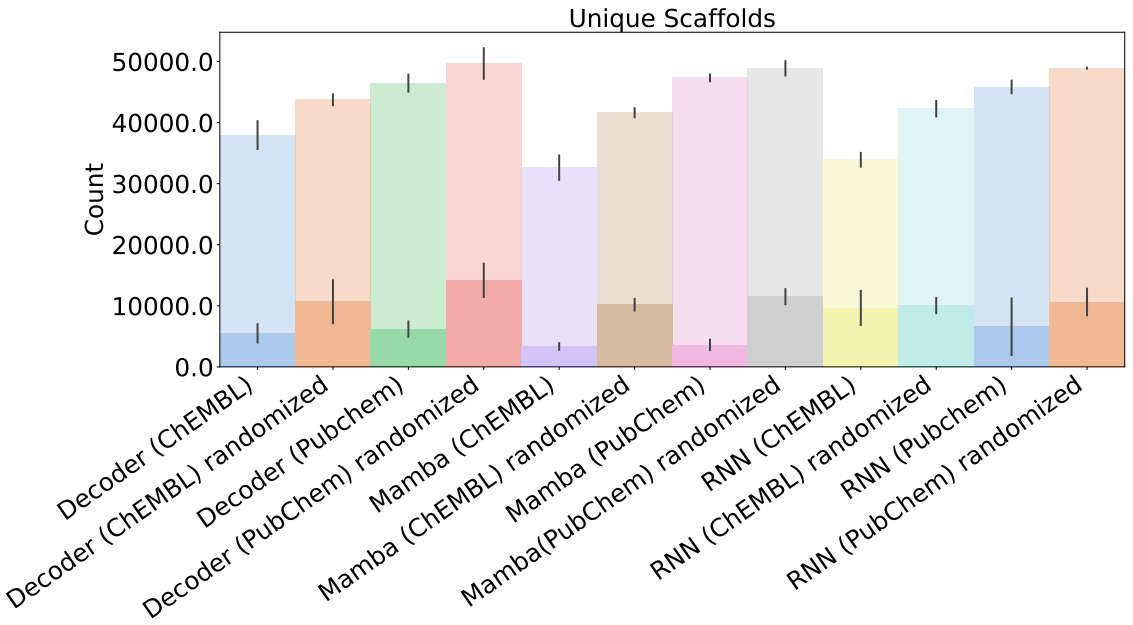


Figure B.12: Number of unique scaffolds generated by all the models pretrained on both datasets with and without randomized SMILES over 1000 steps with a batch size of 64 in RL for target activity (JNK3). The total number of scaffolds is illustrated by the transparent bar, and the solid bar represents the number of scaffolds that also have a score and QED above 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these three runs).

Figure B.13 shows the number of unique generic scaffolds over the 1000 steps per model-dataset pair. The amount of unique generic scaffolds varies between the models. Mamba pretrained on PubChem both with and without randomized SMILES produces relatively many more unique generic scaffolds than the other model-dataset pairs. When including a threshold of a compound also having a score and QED > 0.6, few of the models generate unique generic scaffolds that meet this criteria, but Decoder pretrained on PubChem without randomized SMILES and Mamba pretrained on PubChem with randomized SMILES generate above 5000 unique generic scaffolds over the steps.

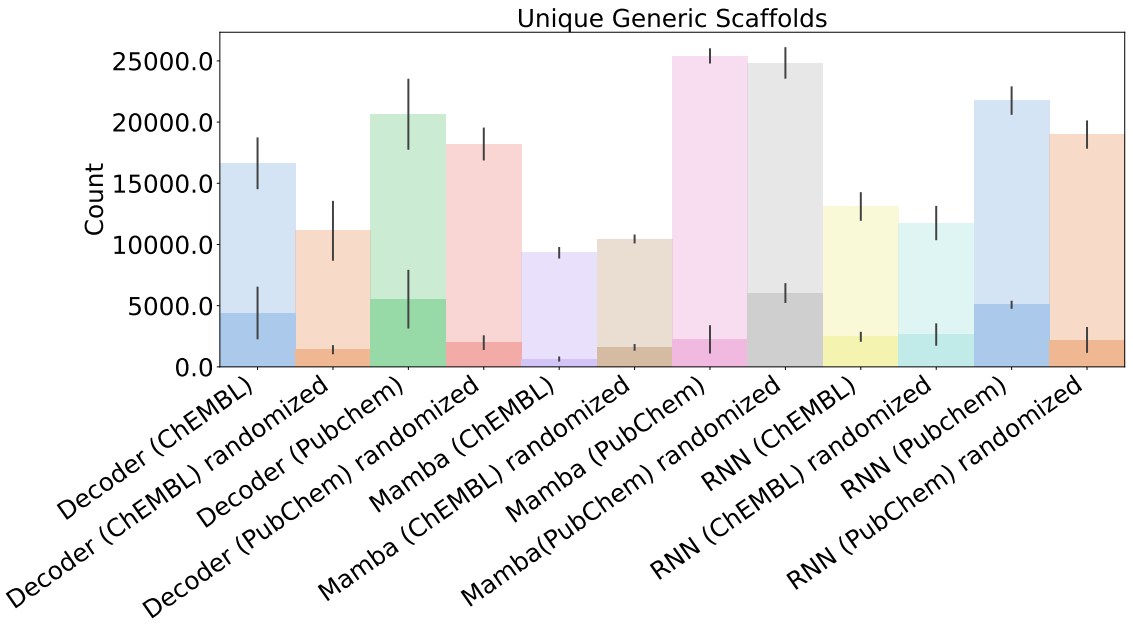


Figure B.13: Number of unique generic scaffolds generated by all the models pretrained on both datasets with and without randomized SMILES over 1000 steps with a batch size of 64 in RL for target activity (JNK3). The total number of scaffolds is illustrated by the transparent bar, and the solid bar represents the number of scaffolds that also have a score and QED over 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these three runs).

B.3.2 GSK3β

To compare how well the models adapted to RL optimization towards activity against the serine/threonine protein kinase GSK3β, a random forest classifier was used as described in Section 3.4. Firstly, the impact from the dataset was evaluated on each of the models, and finally, an evaluation between the three models was made using that model from each architecture that were pretrained on ChEMBL using randomized SMILES. In Figure B.14, the average score and QED over steps in RL for target activity GSK3β for the RNN across datasets are shown.

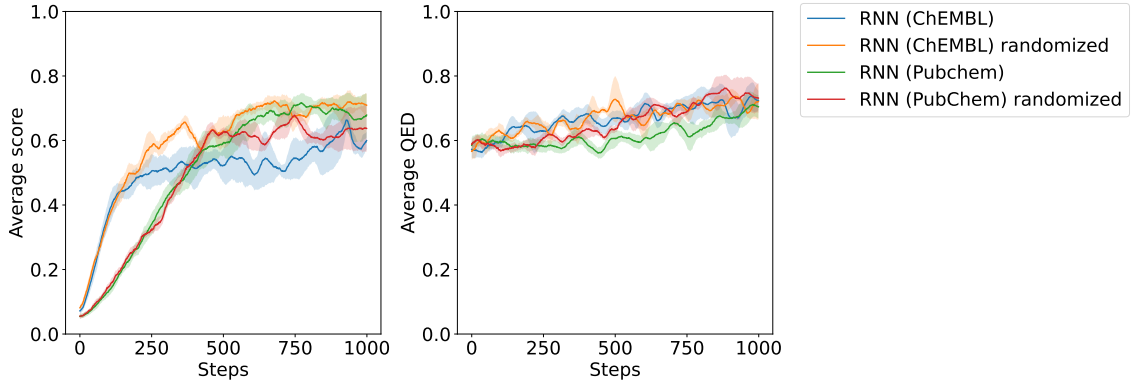


Figure B.14: RNN average score and QED over steps in RL for target activity (GSK3β) across datasets. The models pretrained on ChEMBL adapt faster to the task as compared to the models pretrained on PubChem as seen on the average score converging around 0.6 with slightly lower scores for the PubChem pretrained models. All models adapt similarly to the QED around 0.7.

The RNN pretrained on ChEMBL adapts relatively fast to the task, reaching average scores close to convergence after 250 steps. Those pretrained on PubChem reach those scores after around 500 steps.

In Figure B.15, the average score and QED over steps in RL for target activity GSK3β for the Decoder across datasets are shown.

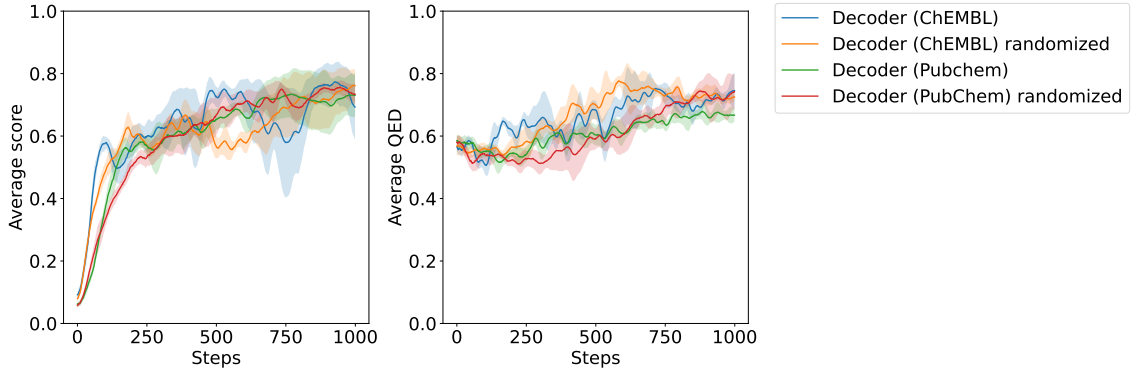


Figure B.15: Decoder average score and QED over steps in RL for target activity (GSK3β) across datasets. All models adapt similarly to the task, finally converging around 0.7. All models adapt similarly to the QED around 0.7.

For the Decoder models, all exhibit fairly similar behavior across the two scoring graphs.

In Figure B.16, the average score and QED over steps in RL for target activity GSK3β for the Mamba across datasets are shown.

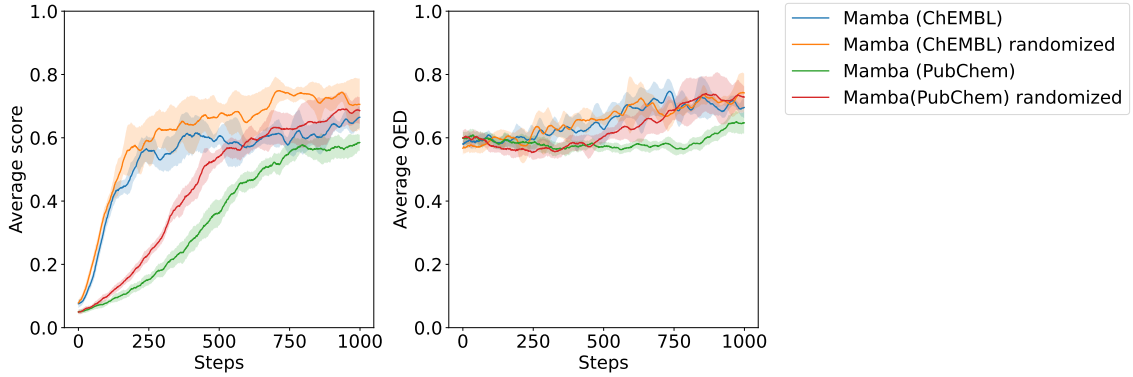


Figure B.16: Mamba average score and QED over steps in RL for target activity (GSK3β) across datasets. The models pretrained on ChEMBL adapt faster to the task as compared to the models pretrained on PubChem as seen on the average score converging around 0.6. All models adapt similarly to the QED around 0.7.

The Mamba models show greater variance in adapting to the task according to the average score. As for the other model architectures, the models pretrained on ChEMBL learns the task during fewer steps than those pretrained on PubChem.

In Figure B.17, the average score and QED over steps in RL for target activity GSK3β for the Decoder, RNN, and Mamba pretrained on ChEMBL randomized SMILES are shown.

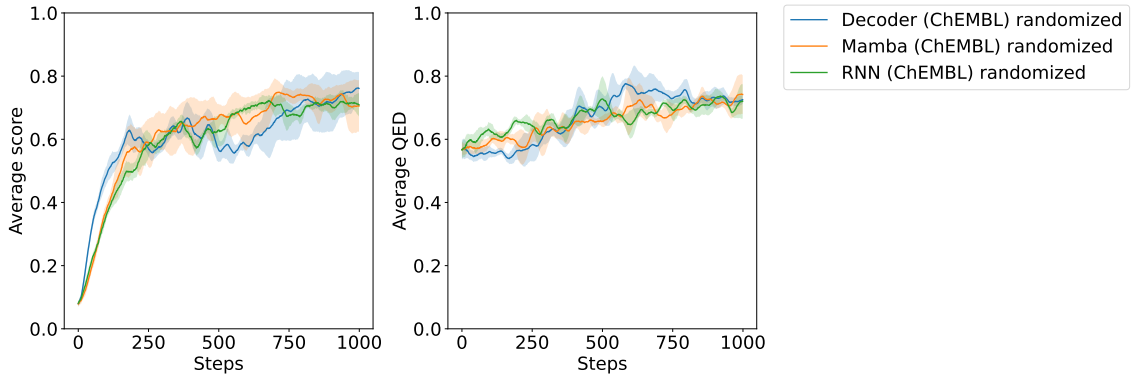


Figure B.17: Decoder, RNN, and Mamba average score and QED over steps in RL for target activity (GSK3β) pretrained on ChEMBL randomized SMILES. The difference in average score and QED is minuscule across the model architectures, all converging at an average score of 0.7, reaching a QED of 0.7 at a similar pace.

When comparing the models pretrained on the same dataset (ChEMBL randomized SMILES), the difference in adapting to the task based on average score and QED is minuscule.

When evaluating the number of unique compounds over the 1000 steps per model-dataset pair, most models generate similar amounts of unique compounds, all generating around 60.000. In Figure B.18, this distribution can be seen. When including

a threshold of a compound also having a score and QED > 0.6, few of their generated unique compounds meet this criterion, which can also be seen in the figures above, for example Figure B.17, since the average score for even the best performing model-dataset pair don't exceed 0.6 with a large margin.

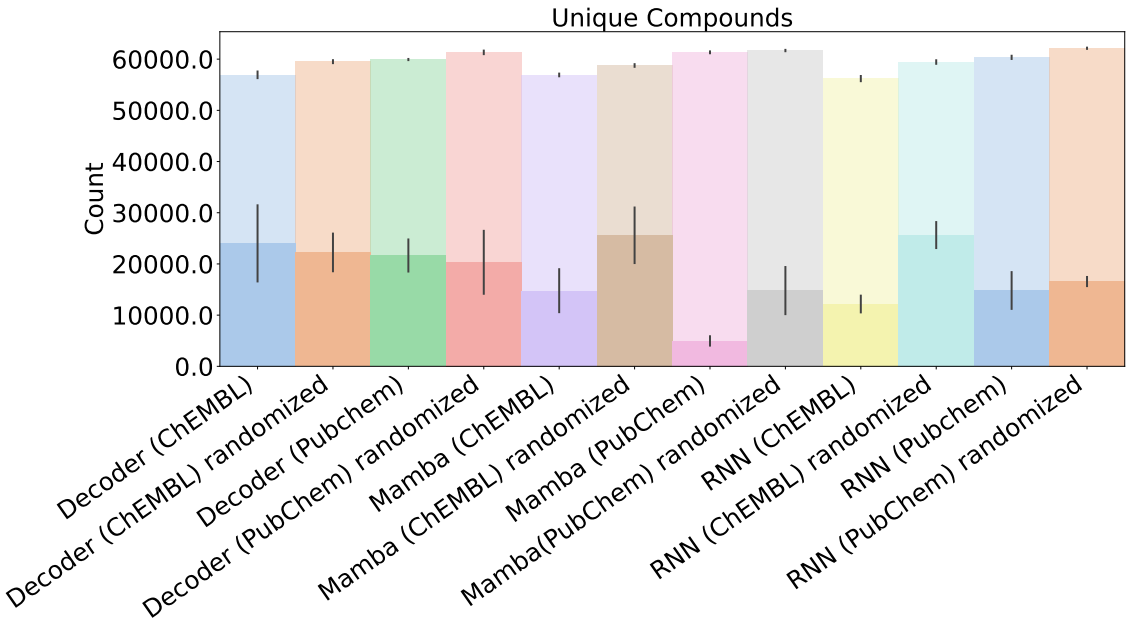


Figure B.18: Number of unique compounds generated by all the models pretrained on both datasets with and without randomized SMILES over 1000 steps with a batch size of 64 in RL for target activity (GSK3β). The total number of compounds is illustrated by the transparent bar, and the solid bar represents the number of compounds that also have a score and QED over 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these three runs).

Figure B.19 shows the number of unique scaffolds over the 1000 steps per model-dataset pair. The amount of unique scaffolds varies between the models. PubChem with randomized SMILES helps the model generate the largest amount of unique scaffolds. When including a threshold of a compound also having a score and QED > 0.6, the behavior varies between the model-dataset pairs.

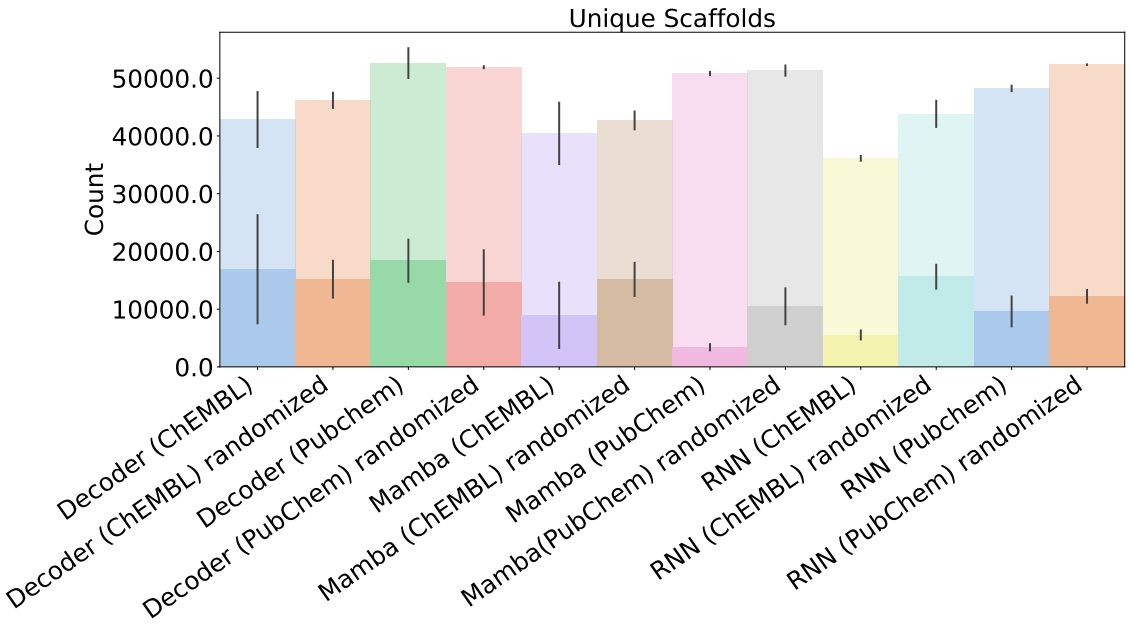


Figure B.19: Number of unique scaffolds generated by all the models pretrained on both datasets with and without randomized SMILES over 1000 steps with a batch size of 64 in RL for target activity (GSK3β). The total number of scaffolds is illustrated by the transparent bar, and the solid bar represents the number of scaffolds that also have a score and QED over 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these three runs).

Figure B.20 shows the number of unique generic scaffolds over the 1000 steps per model-dataset pair. The amount of unique generic scaffolds varies between the models. Mamba pretrained on PubChem both with and without randomized SMILES produces relatively many more unique generic scaffolds than the other model-dataset pairs. When including a threshold of a compound also having a score and QED > 0.6, few of the models generate unique generic scaffolds that meet this criteria, but Decoder pretrained on PubChem with randomized SMILES generates above 5000 unique generic scaffolds over the steps.

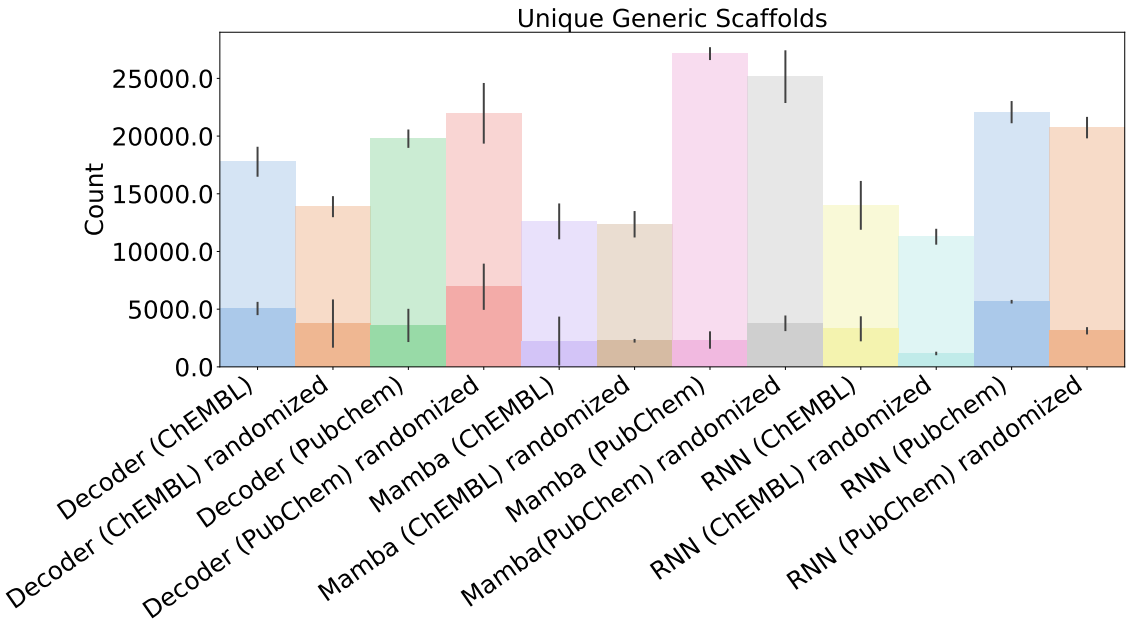


Figure B.20: Number of unique generic scaffolds generated by all the models pretrained on both datasets with and without randomized SMILES over 1000 steps with a batch size of 64 in RL for target activity (GSK3β). The total number of scaffolds is illustrated by the transparent bar, and the solid bar represents the number of scaffolds that also have a score and QED over 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these three runs).

B.4 Additional RL Tasks with the Decoder Mini

In addition to the DRD2 target activity task, the Decoder Mini model pretrained with ChEMBL was further trained for target activity against JNK3 and GSK3β. Moreover, the Decoder Mini was utilized for similarity-guided structure generation with the objective of producing compounds structurally similar to Celecoxib. In the following sections, the results of these reinforcement learning tasks will be presented alongside the corresponding results for the baseline decoder and RNN models pretrained with ChEMBL. For all tasks, QED was added as a scoring function, and the score used to update the models' parameters was the geometric mean over the task-specific score and QED.

B.4.1 JNK3

In Figure B.21, the average target activity score and QED for the Decoder Mini, baseline decoder, and RNN pretrained with ChEMBL when running RL for target activity towards JNK3 can be seen. The initial slope of the target activity score for the Decoder Mini follows that of the baseline decoder and is steeper than that of the RNN. The fluctuations in the target activity scores are large for all three models, with the decoder models being slightly less stable. Large fluctuations are evident in the QED scores as well.

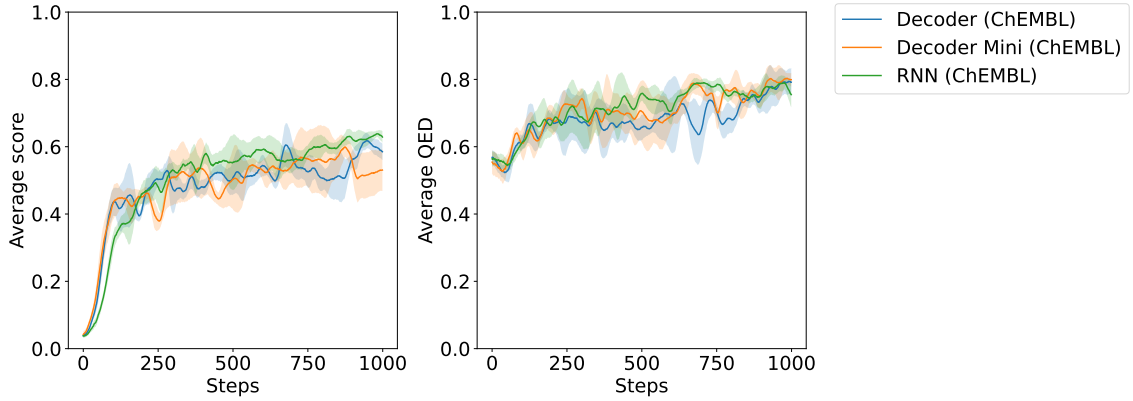
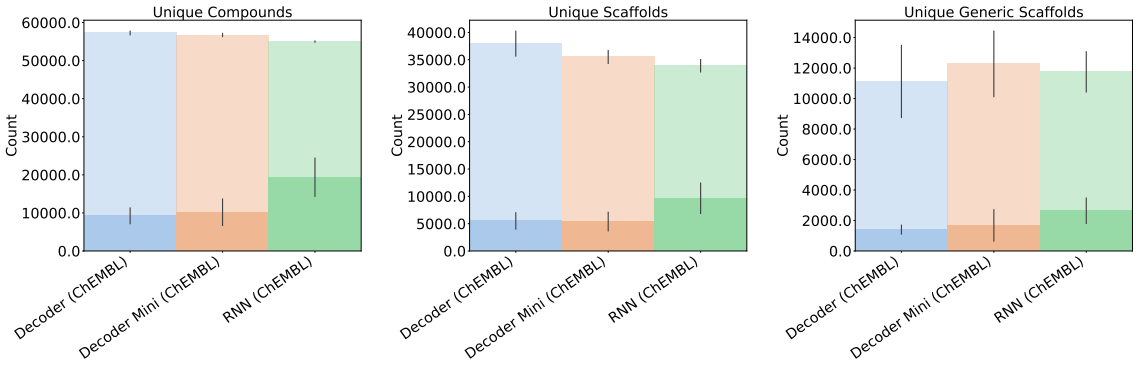


Figure B.21: Per step average target activity (JNK3) and QED score over steps for the baseline decoder, Decoder Mini, and RNN pretrained on ChEMBL (as indicated by legend to the right), over an RL run with DRD2 activity and QED as scoring components. Each line indicates an average over three consecutive RL runs, and the shaded area is the average ± the standard deviation over the three runs.

Studying the diversity among generated compounds, illustrated by the number of unique compounds, unique scaffolds, and unique generic scaffolds in Figure B.22, the Decoder Mini performs similarly to the baseline decoder and the RNN: while providing large diversity overall, few of the compounds have an activity score and QED above the threshold of 0.6. In all diversity aspects, the Decoder Mini matches the performance of the other two models för this optimization task.



(a) (b) (c)

Figure B.22: Number of (a) unique compounds, (b) unique scaffolds, and (c) unique generic scaffolds generated by each model over an RL run of 1000 steps for target activity (JNK3), with a batch size of 64. The total number of compounds/scaffolds is illustrated by the transparent bar, and the solid bar represents the number of compounds/scaffolds that also have a score and QED over 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these 3 runs).

B.4.2 GSK3β

Figure B.23 displays the average target activity score and QED for the Decoder Mini, the baseline decoder, and the RNN models pretrained with ChEMBL during reinforcement learning for target activity against GSK3β. Unlike its baseline counterpart, the Decoder Mini demonstrates less pronounced fluctuations in activity score; however, it does not attain the same maximum activity scores as the baseline decoder. Nevertheless, the activity score of the Decoder Mini remains comparable to that of the RNN. The QED score for the Decoder Mini closely parallels those observed for both the RNN and the baseline decoder.

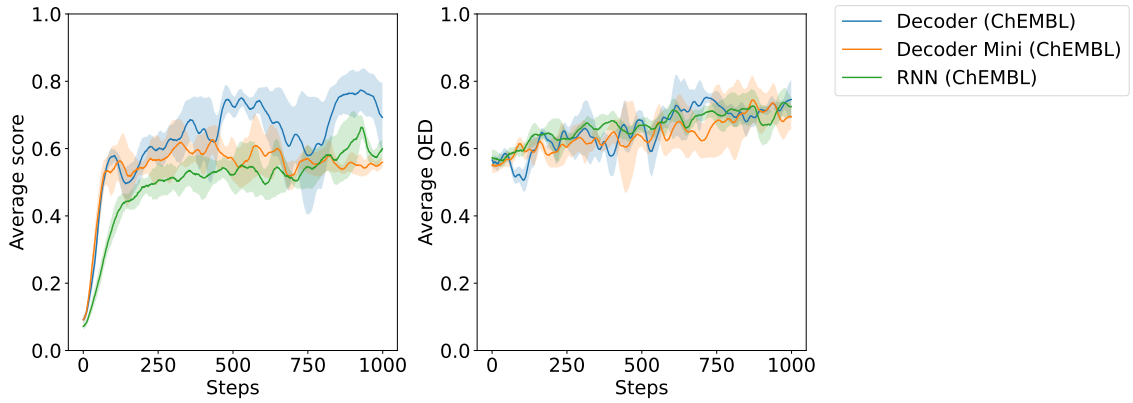
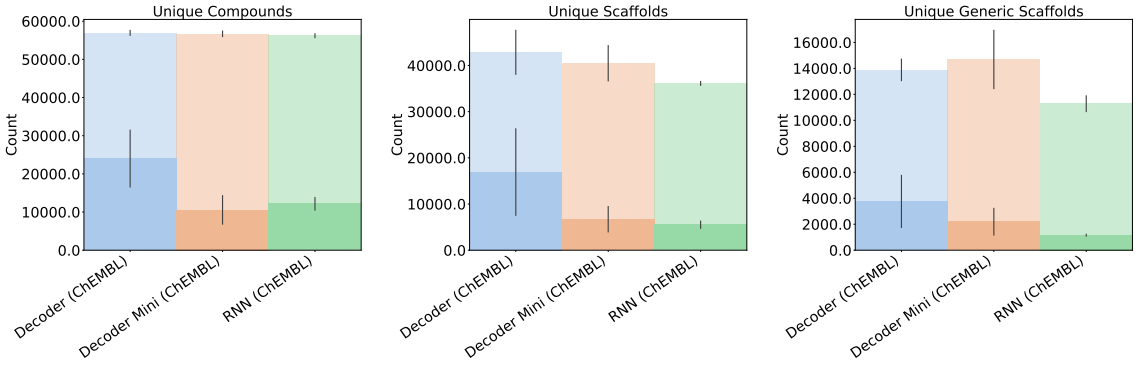


Figure B.23: Per step average target activity (GSK3β) and QED score over steps for the baseline Decoder, Decoder Mini, and RNN pretrained on ChEMBL (as indicated by legend to the right), over an RL run with GSK3β activity and QED as scoring components. Each line indicates an average over three consecutive RL runs, and the shaded area is the average $\pm$ the standard deviation over the three runs.

Regarding the diversity metrics, presented in Figure B.24, the Decoder Mini does not match its baseline counterpart as to generate unique compounds or scaffolds. However, the variations over the different RL runs are large, as indicated by the black lines in Figure B.24.



(a) (b) (c)

Figure B.24: Number of (a) unique compounds, (b) unique scaffolds, and (c) unique generic scaffolds generated by each model over an RL run of 1000 steps for target activity (GSK3β), with a batch size of 64. The total number of compounds/scaffolds is illustrated by the transparent bar, and the solid bar represents the number of compounds/scaffolds that also have a score and QED over 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these 3 runs).

B.4.3 Similarity Guided Structure Generation

The similarity-guided structure generation with the purpose of generating compounds similar to the Celecoxib molecule was done for the Decoder mini as well. The Decoder Mini reached similar similarity and QED scores as the baseline decoder and RNN, see Figure B.25. Similar to the larger decoder, it also exhibited large fluctuations in both scores.

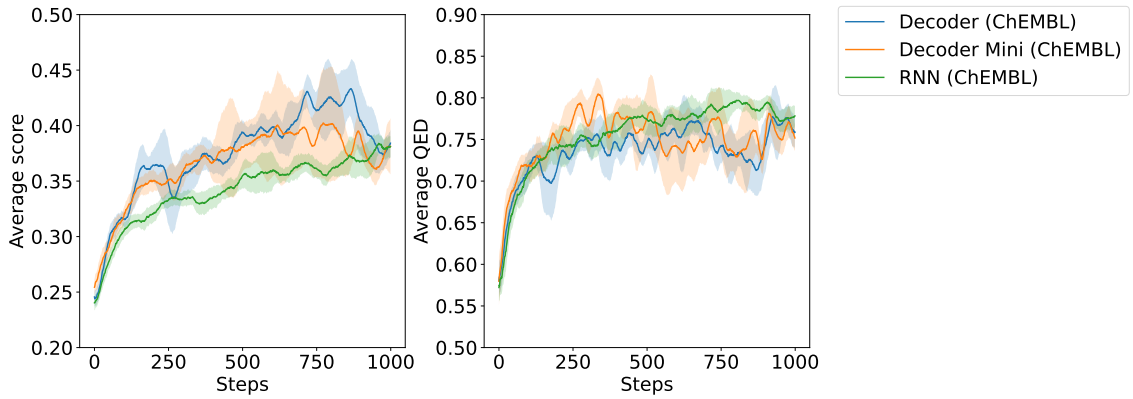
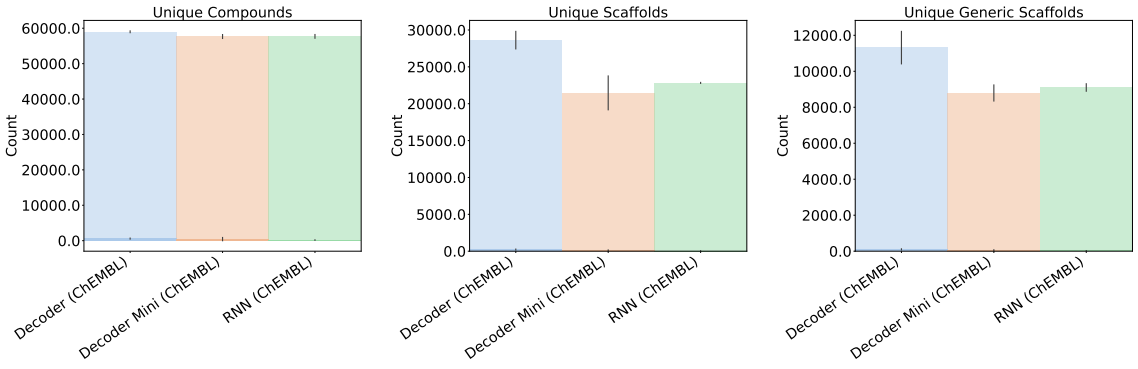


Figure B.25: Per step average similarity score and QED score over steps for the baseline decoder, Decoder Mini, and RNN pretrained on ChEMBL (as indicated by legend to the right), over an RL run for similarity towards Celecoxib and QED as scoring components. Each line indicates an average over three consecutive RL runs, and the shaded area is the average ± the standard deviation over the three runs.

Studying the diversity among generated compounds through the plots in Figure

B.26, the Decoder Mini does not generate as many unique compounds, scaffolds, or generic scaffolds as the larger decoder; however, it does perform equivalent to the RNN.



(a) (b) (c)

Figure B.26: Number of (a) unique compounds, (b) unique scaffolds, and (c) unique generic scaffolds generated by each model over an RL run of 1000 steps for somilarity to Celecoxib, with a batch size of 64. The total number of compounds/scaffolds is illustrated by the transparent bar, and the solid bar represents the number of compounds/scaffolds that also have a score and QED over 0.6. The black lines indicate the standard deviation over three RL runs (with the height of the bar indicating the average over these 3 runs).

B.4.4 Summary: Reinforcement Learning with the Decoder Mini

For both JNK3 and GSK3β, the Decoder Mini displayed a rapid initial improvement in activity score, similar to the baseline decoder, but, as with the baseline decoder, showed marked fluctuations in scores throughout training. Diversity metrics indicated that the Decoder Mini maintained comparable diversity to both the baseline decoder and RNN, although relatively few generated molecules met the higher score and QED thresholds.

On the GSK3β task, the Decoder Mini exhibited less pronounced score fluctuations compared to the baseline decoder but did not reach as high activity scores. In the similarity-guided structure generation task, the Decoder Mini achieved similarity and QED scores close to those of the baseline decoder and RNN, again accompanied by considerable score fluctuations as well as variations over RL runs. While it generated fewer unique compounds and scaffolds than the baseline decoder, its performance in this regard matched that of the RNN.

In summary, the Decoder Mini demonstrates competitive performance in RL-based molecular optimization compared to larger models, particularly in maintaining diversity and achieving reasonable activity and similarity scores despite its reduced parameter count. However, peak score attainment and diversity may be limited compared to the baseline decoder. These findings suggest that the Decoder Mini offers

a parameter-efficient alternative to the larger decoder and RNN, though some trade-offs in optimizability and diversity may need consideration depending on application needs.

C

Summary of Discussion

Table C.1: Summary of key findings on model architecture, training data, and SMILES representations

Factor	Comparison	Observations
Architecture Choice	Decoder-only Transformer, RNN, Mamba, Decoder Mini. Fixed dataset. Pretraining.	Decoder takes the longest time per epoch but reaches the highest validity and uniqueness with a fixed dataset (except for randomized PubChem, where Mamba reaches the highest). Mamba reaches the highest novelty with fixed datasets (except for randomized PubChem, where the decoder reaches the highest. Mamba and RNN converge faster but overfit on ChEMBL. Validity, uniqueness, and novelty differences diminish with a larger dataset (PubChem). Decoder Mini (2.2M params) reaches competitive validity, uniqueness, and novelty, generalizes well, and is faster to train compared to all models.
Training Data	ChEMBL (2.2 M) vs. PubChem (119 M)	ChEMBL leads to faster learning and higher scores, especially in RL tasks, and favors drug-likeness. PubChem grants higher diversity without a threshold, but ChEMBL grants higher diversity with a threshold. Overfitting is less likely on PubChem due to its size. PubChem is significantly slower to train.

Table C.2: Continuation of summary of key findings on model architecture, training data, and SMILES representations

Factor	Comparison	Observations
SMILES Representation	Canonical vs. Randomized	Randomized SMILES in pretraining enhances novelty across models and helps Mamba resist overfitting and improve RL scores. For RNN and the decoder, diversity increases only modestly. Randomized SMILES are, more often than not, preferable in RL adaptation pace. Canonical SMILES can lead to early memorization of training data.
RL Task: Sulphur Avoidance	All architectures	All models learn to avoid sulphur rapidly. Mamba pretrained on PubChem achieves highest generic scaffold diversity above QED and score threshold.
RL Task: Similarity-guided Generation	All architectures	Decoder model adapts quickly, achieves high similarity scores; all models generate many unique compounds, but decoder and RNN trained on randomized SMILES outperform the others in the amount of unique scaffolds. The most diverse dataset, randomized PubChem, performs the worst in score and QED for most models except Mamba.
RL Task: Target Activity (DRD2)	All architectures	No substantial differences in scores across architectures for DRD2. All models learn faster using ChEMBL. All models trained och randomized ChEMBL generate the highest amount of unique compounds meeting the threshold. Randomized ChEMBL makes each model generate the largest number of unique scaffolds.