Transformers: Efficient one-to-many sequence generation

An Encoder Sphere Projection Strategy

Master's thesis in Computer science and engineering

Oscar Almström
Anton Söfting

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Transformers: Efficient one-to-many sequence generation

An Encoder Sphere Projection Strategy

Oscar Almström
Anton Söfting

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Transformers: Efficient one-to-many sequence generation
An Encoder Sphere Projection Strategy
Oscar Almström
Anton Söfting

Supervisor: Nikela Papadopoulou, Department of Computer Science and Engineering
Advisor: Alessandro Tibo, AstraZeneca
Examiner: Miquel Pericàs, Department of Computer Science and Engineering

Master's Thesis 2023
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 2023

Transformers: Efficient one-to-many sequence generation
An Encoder Sphere Projection Strategy
Oscar Almström
Anton Söfting
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Transformers have revolutionized sequence-to-sequence data processing in a wide range of industries. Their ability to handle long-range dependencies and capture contextual information has led to remarkable advancements in speech recognition, image generation, and machine translation. However, current applications primarily focus on one-to-one sequence generation, where a single source sequence is used to produce a single target sequence. In this thesis, we address the challenge of one-to-many sequence generation, where a single source sequence is used to generate multiple target sequences at an architectural level.

To expand the capabilities of the transformer model, we introduce an encoder sphere projection strategy, allowing for scalable and efficient architecture-level variation during sequence generation.

By generating independent vectors with uniform norms and distance from each other, the single source embedding is replicated with an added controlled variation. This expansion enables the shift from a single encoder-decoder relation to a one-to-many batched decoder supporting a set of targets to be processed with the teacher forcing framework.

For the now set-based training, we incorporate a Sinkhorn loss function which encourages variation among generated output sequences while maintaining similarity to the expected targets. The loss calculation involves a pair-wise negative log-likelihood between each predicted output sequence and the ground truth targets associated with the source.

This new architecture supports inherent auto-regressive inference for varied sequence generation, with up to 256 predictions per given source (limited by the model dimension). Compared with the sampling options multinomial and beam search for the base model, the expanded model achieved competitive accuracy and sped up the inference and training time.

We observed a 31% reduced time to train the new model on a single K80 12GB, while a V100 32GB card saw a 27% reduction. The advantages diminished when the overhead of using multiple GPUs was introduced. The inference also showed benefits, reducing the execution time by 9% and 33% compared to multinomial and beam search sampling respectively.

Keywords: computer science, transformer, machine learning, sequence generation, NLP, positional encoding, spherical projection, sinkhorn algorithm, drug discovery, HPC.

Acknowledgements

We are sincerely thankful to Alessandro Tibo, our advisor at AstraZeneca, for introducing us to the assignment and offering continuous support and guidance from the beginning to the end of this thesis. The gratitude expands to the Molecular AI team at the Mölndal AstraZeneca site for welcoming us during our stay and lending us the hardware utilized throughout this thesis. Additionally, we would like to thank Nikela Papadopoulou, our supervisor at Chalmers, for providing valuable feedback, suggestions and room for discussion to push this research forward. Finally, we would like to thank Miquel Pericàs, our examiner, who provided valuable comments and feedback.

Oscar Almström and Anton Söfting, Gothenburg, 2023-06-19

Contents

List of Figures

List of Tables

1 Introduction

Transformers[1] have revolutionized the way we process sequence-to-sequence (seq2seq) data in a wide range of industries [2]–[8]. The versatility and power of the transformer is evident in several successful applications since its introduction by Vaswani et al.[1]. For instance, in speech recognition, transformers have achieved state-of-the-art performance by leveraging their ability to handle long-range dependencies and capture contextual information[1] while only requiring a fraction of the training cost compared to recurrent models [9]. In image generation, transformers have shown remarkable results in tasks such as generating realistic images from textual descriptions [10]. In machine translation, transformers have outperformed traditional state-of-the-art recurrent neural network approaches and have even been used to create real-time translation tools [11]. These examples demonstrate the incredible potential of transformer models in a variety of contexts, with the global phenomenon ChatGPT[12] stemming from research on generative pre-training language transformers[13] currently spearheading its way to the forefront of AI discussion.

The key features of the transformer that enabled its breakthrough include the ability to effectively capture long-term dependencies within a sequence. Transformers consider the entire input sequence as a whole, allowing them to capture relationships between different elements, such as the subject and object, across sentences.

Another compelling feature is support for flexible sequence lengths, achieved by employing padding techniques to handle variable lengths, while positional encoding enables effective translation across languages with varied sentence structures. Furthermore, transformers are more efficient than traditional recurrent neural networks due to their attention mechanism, which can process inputs by handling word similarity and context in parallel. This efficiency also enables large-scale models to better utilize the computational power present in high-performance computing platforms commonly facilitated for machine learning.

In this thesis, we build on the groundbreaking work of the base transformer model in natural language processing and explore ways to expand its capabilities. There are still challenges to be addressed with the transformer to fully leverage the power of these models. In particular, we focus on the problem of one-to-many sequence generation, where a single source sequence is used to generate multiple target sequences. Current applications are designed to favor one-to-one sequence generation, such as in language text translation where the model returns the most probable and best match at all times. A deterministic translation could be seen as the optimal model in this scenario, however, there are cases where many predictions are a better suit and diversity is beneficial [14]–[16]. This is an important and challenging problem that has broad applications in machine learning and natural language processing.

This thesis contributes to transformer research by exploring the expansion of the transformer architecture to generate many sequences from a single source. More precisely, this is pursued by modifying the standard encoding-decoding process of the transformer through the introduction of an encoder sphere projection strategy. We aim that such a strategy enables architecture level variation and thus diverse generative sampling while opening up avenues for batched parallel and thus efficient decoder computation.

1.1 Thesis Outlook

This thesis will first introduce the relevant background knowledge to understand the transformer architecture and problem at hand. Next, we cover the dataset used and the necessary preprocessing steps. After that, the method chapter will describe our proposed one-to-many transformer architecture with the decisions that were made and how we developed and evaluate the model. This is followed by the results that present a comprehensive evaluation of our model, analyzing metrics and providing insights into its performance compared to the base model. Finally, the discussion and conclusion chapter interprets the results, discusses challenges during the development, concludes the research as a whole, and proposes future research directions.

2

Background

In this chapter, the necessary background knowledge required to understand the problem and solution is presented. Section 2.1: Artificial Neural Networks, introduces the topics Neural networks and how this ties into High-performance computer systems and Natural language processing (NLP). Section 2.2: Transformers, describes a transformer with the components that build a transformer. Section 2.3: Sinkhorn Distance, explains the algorithm and how it is used to regulate the attention weights. Lastly, Section 2.4: Inference and Sampling, describes how the inference and sampling works for the transformer.

2.1 Artificial Neural Networks

The topic of AI and ML often refers to advanced techniques based on artificial neural networks (ANNs) due to their ability to learn and adapt to different patterns and data types. Performing complex computations that previously were impossible for traditional algorithms can be attributed to how ANNs are modeled after the structure and function of the human brain, mimicking the synapses between neurons.

At the core of an ANN is a network of interconnected nodes or neurons that process and transmit information through weighted connections. Each neuron receives input from other neurons, applies an activation function to the input, and then transmits the output forward to other connected neurons further ahead in the network (feed-forward network). Typically a nonlinear activation function is used to introduce nonlinearity to the network which enables it to learn more complex patterns [17].

Training a neural network involves adjusting the weights, being the parameters of the ANN, between its nodes to minimize a loss function that measures the difference between the predicted output of the network and the actual output. This process of adjusting the weights is known as backpropagation and refers to the calculation of the gradient of the loss function with respect to the set of its weights. The calculation can be computationally expensive for large networks due to the abundance of matrix multiplications involved, often resulting in long training time even for dedicated hardware. The model is typically trained until it reaches a point where the loss function no longer decreases without over-fitting to the training data. Over-fitting occurs when the model becomes too specialized in its predictions for the given training data and fails to generalize to new, unseen data. At this point, the model may start to memorize the training data instead of learning the underlying patterns.

Once the model has been trained to a satisfactory level, it can then be used to make accurate predictions on new input data of the same type.

Deep neural networks (DNNs) are expansions on ANNs and a cornerstone in modern ML. They consist of multiple layers of interconnected nodes (referred to as hidden layers), where each layer processes the output of the previous layer. Input data is fed to the first layer, and processed through the many connected hidden layers before an output layer produces the final output. The hidden layers are where the connections and activation functions can differ heavily between models and enable task-specific variation. Different patterns of intra-layered connections can allow the DNN to perform different tasks, such as computer vision[18], natural language processing[19], and classification[20]. For example, a convolutional neural network (CNN) is designed for image recognition tasks, while a recurrent neural network is designed for sequential data such as speech or text. There exist two main variations of DNNs, either feed forward meaning the information only travels from the input node towards the output and recurrent with feedback loops that reuse previous partially or completed outputs.

Regardless of model shape, a common denominator for all ML models is the cumbersome task of training them. The success of deep learning (DL) is contributed to the ability to scale by adding more hidden neurons. The additional neurons increase the number of parameters and thus the model complexity, which from a statistical machine learning standpoint has the potential of lowering the bias and variance [21]. Furthermore, the model requires more neurons to discriminate varied patterns in the data, but a balance need to be struck. Too few neurons hinder the capturing of all relevant information in the input data, while too many are prone to resulting in overfitting [22]. Complex models require careful selection of many hyper-parameters such as weight decay, learning rate, momentum and learning decay, not to mention the activation functions and layer structure. Achieving acceptable results for bigger models require larger datasets and longer training, meaning the host system containing the model is put under more strain [23]. It is customary to employ HPC platforms to manage the ML training (and in some cases inference too) as it is often the only feasible way to complete the task within a reasonable time-frame with regards to computational and memory demands [24].

HPC systems in the realm of ML usually consist of clusters of Graphics Processing Units (GPUs) that are optimized for accelerating the matrix multiplication present during training. The ability of GPUs to perform parallel computations has made them an attractive option for training large neural networks. By using multiple GPUs in a cluster, the computation time can be significantly reduced. In addition, the high memory bandwidth of GPUs enables efficient computation of vector-matrix or matrix-matrix multiplication, which is a key operation in deep learning [21]. The throughput-oriented architecture of a GPU with high memory bandwidth is a field studied thoroughly as the optimal algorithm differs depending on memory constraints[25], matrix-sparseness[26], precision[27] and level of distribution [28]. With the introduction of software frameworks such as *Compute Unified Device Architecture* (CUDA) one can at a simple C++ language-level program efficient code for GPU offloading that combines both multi-core central processing unit (CPU) and accelerators for ANN computation of all shapes [29]. There now exists highly accessible Python packages that leverage GPU offloading through CUDA such as PyTorch [30], allowing users to focus on the architecture and layout of the ML models rather than computation.

2.1.1 Natural Language Processing & Recurrent Neural Networks

Natural Language Processing (NLP) is a subfield of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and language. The goal of NLP is to enable computers to understand, interpret, and generate language. NLP involves a wide range of tasks, including but not limited to language understanding, sentiment analysis, machine translation, and text generation. Moreover, NLP often involves preprocessing steps such as tokenization, stemming, and lemmatization to transform the raw text into a format that can be easily processed by machine learning algorithms.

Recurrent Neural Networks (RNNs) are a type of neural network that is particularly useful for processing sequences of data and in this case, text. Unlike feedforward neural networks, which process each input independently, RNNs have a "memory" that allows them to take into account the previous inputs in a sequence. The basic architecture of an RNN consists of a recurrent layer, which passes information from one time step to the next, and a set of weights that determine how information is combined across time steps. One of the main advantages of RNNs is their ability to handle variable-length sequences, as the same weights can be applied to inputs of different lengths [31].

RNNs have been applied to a wide range of NLP tasks, including language modeling, machine translation[1], sentiment analysis[32], and speech recognition[33]. They are particularly well-suited for modeling sequential data due to their ability to maintain a memory of previous inputs. RNNs are trained using backpropagation through time which involves unfolding the network and applying the standard backpropagation algorithm to update the weights.

Furthermore, more complex variants of RNNs have been developed, such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), which have been shown to be particularly effective at modeling long-term dependencies in sequences. LSTM networks use gating mechanisms to selectively remember or forget information from previous time steps, while GRUs use gating mechanisms to control the flow of information and prevent the vanishing or exploding gradient problem. These advanced RNN architectures have been successfully applied to various NLP tasks and have significantly improved performance compared to traditional RNNs.

2.1.2 Long Short-Term Memory

Long Short-Term Memory (LSTM) networks are a type of RNN that can learn long-term dependencies in sequential data. LSTMs are designed to remember information over a period of time, which makes them well-suited for tasks with patterns in data sequences such as language modeling, speech recognition, and machine translation [34]. In an LSTM, each unit contains a memory cell, an input gate, an output gate, and a forget gate. These gates control the flow of information into and out of the memory cell, allowing the LSTM to selectively remember or forget information [35]. The memory cell stores the previous state, which can be modified by the input gate and the forget gate. The input gate determines which information is relevant and should be stored in the memory cell, while the forget gate decides which information should be removed from the memory cell. The output gate then controls the flow of information out of the memory cell, which is the output of the LSTM unit.

In an LSTM, each time step in the sequence is processed sequentially, with the output from one time step being used as input to the next time step. This sequential nature makes it difficult but not impossible to parallelize the computation across multiple processors or GPUs [36]. They can however be stacked on top of each other to form a deep LSTM network, which has been shown to be effective in modeling complex sequential data.

2.2 Transformers

Transformers is a revolutionary feed-forward neural network architecture that has garnered widespread attention and adoption in NLP. The original concept was introduced by A. Vaswani, et al. in the seminar paper "Attention is all you need," [1] and offers several architectural advantages over traditional NLP models such as RNN. The transformer architecture is the first transduction model relying entirely on self-attention without using sequence-aligned RNNs or convolutions to compute representations of its input and output.

The transformer architecture has an encoder-decoder structure as most competitive neural sequence transduction models[37], [38], each composed of multiple layers of self-attention and feed-forward neural networks as seen in Figure 2.1. The encoder takes a sequence of input tokens and generates a sequence of encoded representations, while the decoder takes these representations and generates a sequence of output tokens [39].

The output from the decoder is then passed through a fully connected layer that is used to transform the encoded input sequence into a sequence of hidden states. These hidden states are then used as input to the subsequent layers of the decoder. This is followed by a softmax layer that takes a vector of inputs and normalizes them such that the sum of the outputs is equal to one. Each output of the softmax layer represents the probability of the input belonging to a particular class. In short, the softmax layer produces a probability distribution over the vocabulary of possible words for each time step in the decoder.

Attention is a key component of the transformer architecture and is used to compute representations of the input sequence. In the context of transformers, an attention function can be described as mapping a query vector and a set of key-value pairs, resulting in an output vector. In this function, the query, keys, values, and output are represented as vectors. The output vector is obtained by computing a weighted sum of the values, where each weight is determined by a compatibility function that evaluates the similarity between the query and the corresponding key [1].

The self-attention mechanism is the core of the transformer architecture. It allows the model to attend to different parts of the input sequence during computation, enabling it to capture long-range dependencies and handle variable-length input sequences. In addition to self-attention, the transformer architecture employs residual connections and layer normalization to improve training stability and performance. The architecture is also highly parallelizable, allowing for efficient training on large-scale datasets.

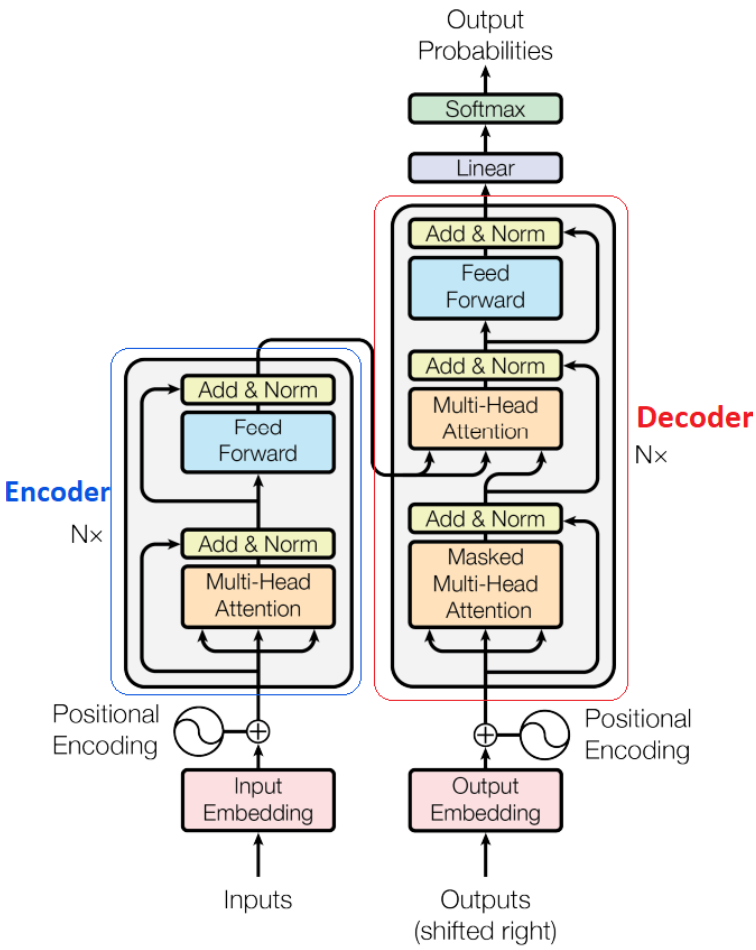

Figure 2.1: An overview of the transformer architecture, from "Attention is all you need" by A. Vaswani, et al.[1]

2.2.1 Attention

Attention mechanisms are powerful components in machine learning that enable models to assign varying degrees of importance to different parts of the input sequence during the prediction process. By leveraging attention, models can focus on relevant information and capture dependencies between different parts of the input, leading to improved performance.

One specific type of attention is self-attention, also known as intra-attention, which targets the same input sequence. Self-attention has proven to be particularly effective for tasks involving long sequences, such as machine translation[1] and summarization[40]. By allowing the model to capture and evaluate the relationships between different parts of the same input, self-attention enhances the model's ability to make accurate predictions.

The fundamental idea behind self-attention is to compute a weighted sum of the same input sequence, where the weights are learned dynamically based on the content of the input. For each position in the input sequence, the self-attention mechanism computes a vector of weights that reflects the importance of the other positions with respect to the current position for the current task. An example of this is seen in Figure 2.2 where the translation in red shows the direct translation and the one in green shows the result after implementing weights and how the order of the words are changed with said weights.

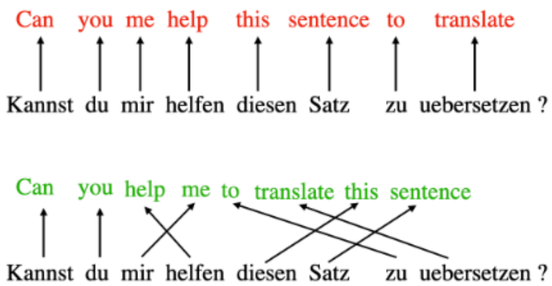

Figure 2.2: The self-attention mechanism for translation from German to English [41].

Two commonly used attention functions in neural networks for natural language processing tasks are additive attention and dot-product attention. Additive attention is a method that learns a weighted combination of the input features using a neural network with a single hidden layer. In contrast, dot-product attention computes the attention weights as the dot product between a query vector and a set of key vectors. While both methods are effective in capturing the relevant information in the input sequence, dot-product attention is much more space-efficient and has the ability to capture complex relationships between query and key vectors [1].

Notably, self-attention has proven to be particularly effective when used in conjunction with the transformer architecture. The transformer model consists of a series of stacked self-attention layers, each of which is followed by a feed-forward neural network. The self-attention layers allow the model to selectively attend to different parts of the input sequence, while the feed-forward layers provide a mechanism for non-linear transformations of the representations.

Self-attention has emerged as a powerful tool for NLP tasks, allowing models to capture complex relationships within input sequences and achieve state-of-the-art performance in a variety of domains such as vision and speech [42].

2.2.2 Multi-Head Attention

Multi-head attention is a mechanism in deep learning that allows a model to attend to different parts of the input sequence or representation in parallel using multiple attention heads, allowing it to capture more complex relationships and patterns compared to using a single head. In multi-head attention, it computes multiple attention scores for each token using different linear projections of the input embeddings. These attention scores are then concatenated and passed through another linear projection to produce the final values, as depicted in Figure 2.3 [1].

The idea behind multi-head attention is to allow the model to attend to different aspects of the same input sequence simultaneously by using many attention layers in parallel. For example, one attention head may focus on syntactic relationships between words, while another attention head may focus on semantic relationships. By computing multiple attention scores in parallel, the model can capture a more comprehensive representation of the input sequence and enhance its performance on a wide range of natural language processing tasks.

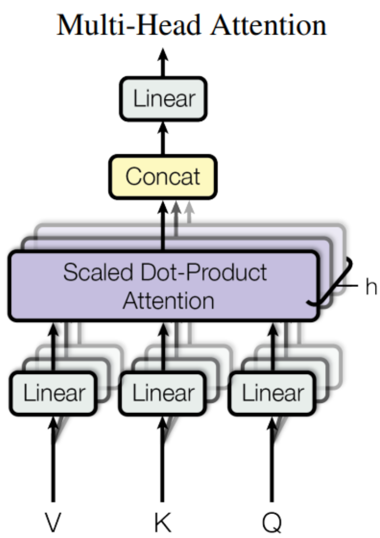

Figure 2.3: Model of multi-head attention architecture, from "Attention is all you need" by A. Vaswani, et al.[1]

2.2.3 Encoder and Decoder

The encoder component in the transformer architecture is comprised of a stack of multiple layers that are structurally identical, as seen in Figure 2.4 where two layers are in place. Each layer contains two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. To facilitate the residual connections necessary for efficient training, each sub-layer output is normalized and added to the input via a residual connection, which is then followed by layer normalization [1]. These residuals address the vanishing gradient issue by allowing earlier input to also bypass sublayers, thus increasing the flow of gradient information during training. As mentioned, the encoder takes a sequence of input tokens and processes them to generate a sequence of encoded representations. Each input token is embedded, and positional encodings are added to incorporate positional information. The encoded representations capture the contextual information of the input sequence

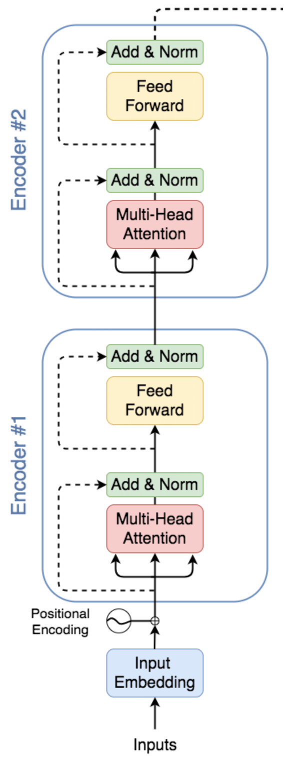

Figure 2.4: Overview model of the encoder architecture, from "Attention is all you need" by A. Vaswani, et al.[1]

The decoder component is also comprised of a stack of multiple identical layers. Each layer consists of three sub-layers, including the multi-head self-attention mechanism and position-wise fully connected feed-forward network present in the encoder layers, as well as an additional sub-layer that performs multi-head attention over the output of the encoder as seen in Figure 2.5. As in the encoder, residual connections are employed around each of the sub-layers, followed by layer normalization.

To prevent positions in the self-attention sub-layer of the decoder pipeline from attending to subsequent positions, subsequent masking is applied. Additionally, the output embeddings are offset by one position, ensuring that the predictions for position i rely solely on the known outputs at positions less than i. These modifications to the self-attention sub-layer and output embeddings enable the decoder to generate accurate and coherent output sequences based on the encoded representations generated by the encoder [1].

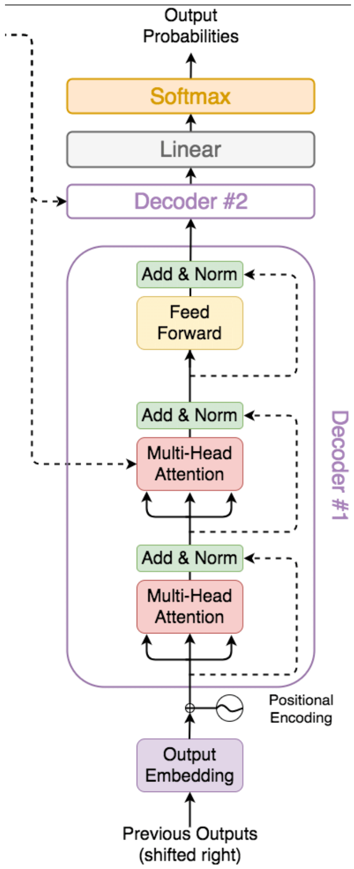

Figure 2.5: Overview model of the decoder architecture, from "Attention is all you need" by A. Vaswani, et al.[1]

An important aspect to note, as seen from the dotted arrows coming from the left in Figure 2.5, is that the encoded embedding is added in parallel at a later stage of the decoder. The feature embedding tied to the source is not required for the decoder to function and instead acts as additional input, referred to as feature forcing [43]. The output embedding depicted at the bottom of the same figure is the feature vector of the ground truth target in question, which is known during training. The training is not autoregressive in the sequence generation, instead, a teacher forcing framework is applied to ensure the correct starting sub-sequences are always used [44]. As such, all possible sub-sequences for the model to learn are available from the start which combined with the encoder embedding allows the model to in parallel process all output probabilities over the entire sub-sequence space. The generative nature aims to capture the next token in the sequence given all previous tokens available, as depicted in a short example in Equation 2.1 (with SMILES rather than the actual feature representations of the target embeddings).

$$\begin{aligned}
&\text{Target} &&: [1, C, H, C, C, C, O, C, Br, O, H, C, 2] \\
&\text{SubSeq}_1 &&: [1] \rightarrow [C] \\
&\text{SubSeq}_2 &&: [1, C] \rightarrow [H] \\
&\text{SubSeq}_3 &&: [1, C, H] \rightarrow [C] \\
&... && \\
&\text{SubSeq}_n &&: [1, C, H, C, C, C, O, C, Br, O, H, C] \rightarrow [2]
\end{aligned} \tag{2.1}$$

Here, the target is subsequently masked to isolate the part of the sequence relation to learn to predict in the current forward pass of the decoder. With contribution from the source embedding the model can compute the output probabilities in parallel and eventually learn what the most probable next token is given only a source and the previous tokens predicted.

2.2.4 Positional Encoding

Positional Encoding (PE) is a technique used in transformer architectures to incorporate positional information into input embeddings of a sequence. Unlike recurrent neural networks, transformers process the entire sequence at once using self-attention, meaning the notion of element order is lost [1]. Each word is fed in parallel to the encoder as the sequence is processed in an independent and isolated environment, all with the same underlying model weights however. The ordering is crucial for sentence cohesion and relations between words based on relative position which means this information needs to be explicitly added to the input sequence. A naive approach is to assign each word a number linearly, meaning the first word is labeled "1", "2", "3" and so on. Complications with this however is scaling with longer sentences, especially when the model is introduced to sentences longer than trained on [45]. It also introduces a higher intensity to the appended encoding as the index number increases which risks skewed generation.

The desired attributes of the positional encoding are to assign unique, uniform and independent values to each token in a deterministic and scalable way. The proposed solution by Vaswani et al.[1] was to provide this information by adding a fixed sinusoidal function of the token position to the input embedding vector. This injects the order of words into the model's input through simple vector addition rather than index association. Let the positional encoding function be $f : \mathbb{N} \rightarrow \mathbb{R}^d$ with d representing the dimensions of the encoding.

$$f(t)^i := \begin{cases} sin(\omega_k \cdot t), \text{ if } i = 2k \\ cos(\omega_k \cdot t), \text{ if } i = 2k + 1 \end{cases} \tag{2.2}$$

where $\omega_k = 10000^{-2k/d}$ and t is the input sequence position with k the interpretation of each dimension of the positional encoding i. In Figure 2.6 the PE matrix is visualized showing how varied intensity is assigned to each index in the sequence across the model dimension.

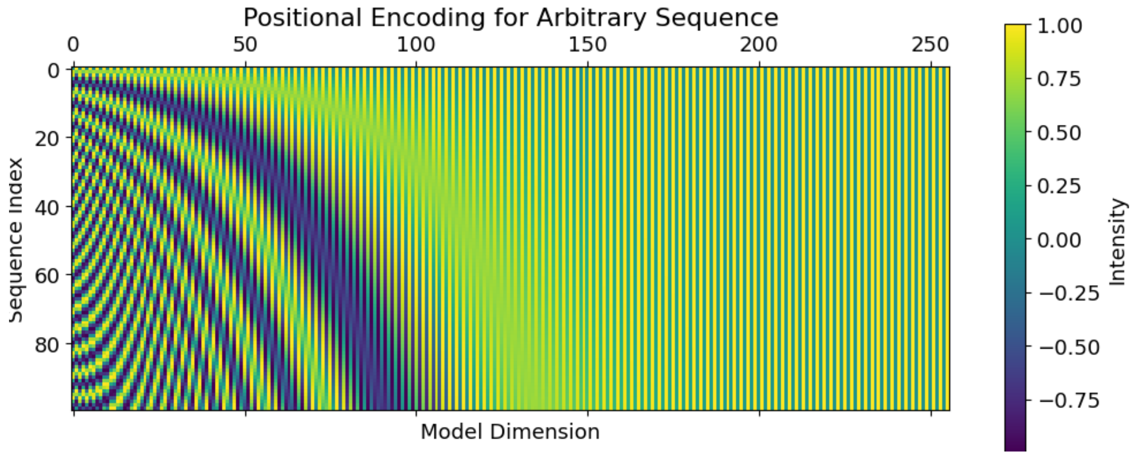

Figure 2.6: Visualization of the Positional Encoding matrix for x-axis $d = 256$ and the y-axis sequence length $k = 100$.

The result is a geometric progression from 2π to $10000 \cdot 2\pi$ with the desired attributes in the form of a vector of the same dimensions as the model for easy addition. This enables parallel encoding and decoding of each token or word in a sequence without losing the information tied to their relative position.

2.3 Sinkhorn Distance and Sinkhorn-Knopp Algorithm

The Sinkhorn distance is a popular distance metric in machine learning and optimization that has gained significant attention in recent years due to its ability to measure the similarity between two probability distributions. The distance is derived from the optimal transport problem which seeks to determine the most efficient way to transport mass from one distribution to another, subject to certain constraints and a cost/distance matrix [46]. A simple example can be seen in Figure 2.7 where the two producers deliver their goods to two consumers.

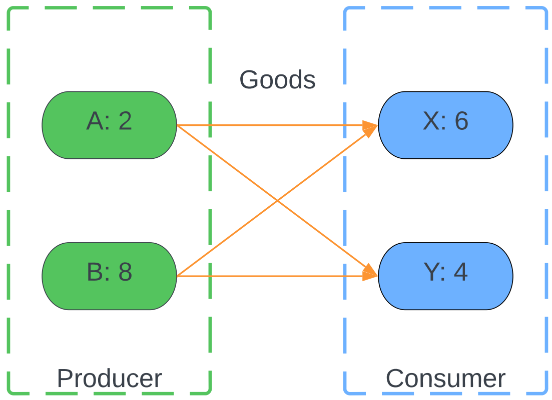

Figure 2.7: Illustration of an Optimal Transport Problem with two producers (left) and two consumers (right).

The non-negative distance matrix would represent the static cost of transporting one unit of the goods from either producer A or B to consumer X or Y, written as $D(i, j) = |i - j|$ with $i \in P = \{A, B\}$ and $j \in C = \{X, Y\}$. The amount of goods being transported is depicted in $T(i, j) \geq 0$ which allows for the minimization of the optimal transport problem as:

$$\sum_{i,j} T(i, j) \cdot D(i, j) \tag{2.3}$$

As the amount of produced and consumed goods are equal one can normalize both and treat them as probability distributions $p(i)$ for the producers and $c(j)$ for the consumers. With these two distributions and the distance matrix D, the total cost of transportation can be seen as a measure of the distance between the two distributions, with respect to the cost relation.

An efficient iterative numerical method of computing the Sinkhorn distance is the Sinkhorn-Knopp algorithm which approximates the optimal solution. Given its iterative property, one can tweak the number of iterations to achieve the desired level of accuracy. In essence, the algorithm starts with two non-negative matrices, P and Q, which represent the two probability distributions that need to be matched. The goal is to find a matrix M that minimizes the cost of matching the two distributions, given a distance cost function D.

The algorithm proceeds iteratively, updating the rows and columns of M to satisfy the following conditions:

- $\sum_i M_{i,j} = P_j$
- $\sum_j M_{i,j} = Q_i$
- $T_d = \min(\sum_i \sum_j M_{i,j} \cdot D_{i,j})$

To satisfy the first condition, the algorithm normalizes the rows of M by dividing each element by the sum of the row. Similarly, the second condition requires one to normalize the columns of M by dividing by the respective sum of the columns. These normalization steps ensure that the row and column sums of M match the target distributions P and Q, respectively. Furthermore, scaling significantly reduces the computational complexity at each step keeping it low throughout the entirety of the approximation.

To satisfy the third condition, minimizing the total distance T_d, the algorithm iteratively updates the elements of M through the following formula:

$$M_{i,j} = P_i \cdot Q_j / (K_i + L_j - D_{i,j}) \tag{2.4}$$

where K and L represent vectors for the row and column sums of the current iteration matrix M, respectively. The algorithm repeats the row and column normalization steps and updates M using the formula above until it converges to a stable solution. Given the abundance of matrix multiplications in this algorithm it can be vectorized and achieve a computational complexity of $O(N^2)$[46] compared to that of EMD at $O(N^3 \cdot logN)$[47] with N transport elements to consider. Provided that the matrix has total support [48], the algorithm will inevitably converge to the optimal stable solution within a finite time. This property renders the algorithm effective for repetitive training procedures [49].

2.4 Inference and Sampling

Generating output is what the transformer is built and trained for, with the purpose of predicting sequences based on new unseen data at inference time. In the example of language translation, each word in the input is translated in order (with the attention mechanism ensuring context and continuity) meaning there are hypothetically millions of possible output sequence combinations. Sampling out the few valid sequences is ultimately a matter of picking the most probable word or token at each iteration step from the output probabilities, seen at the top of Figure 2.5.

There exists plenty of methods to make the selection based on the probabilities depending on what end result one is pursuing and the available resources. A greedy approach is to always pick the most probable token at all times, which often results in a single decent quality output without introducing close to any additional computation. If one desires more variation, multinomial sampling instead makes the selection based on the probabilities thus leveraging the uncertainty of the model prediction. A short pseudocode example can be seen in Algorithm 1, displaying model inference and generation of a sequence with either greedy or multinomial sampling.

Algorithm 1 Auto-regressive Inference Pseudocode

Require: src, $model$, $N > 0$
  memory $\leftarrow model.encode(src)$
  ys $\leftarrow start\_token \cdot N$
  while ys$[-1] \neq end_token$ **do** ▷ Abort when all N sequences end
    out $\leftarrow model.decode(memory, ys)$
    token_prob $\leftarrow model.generator(out)$
    if Greedy **then**
      next_token $\leftarrow$ max(token_prob)
    else if Multinomial **then**
      next_token $\leftarrow$ $random$(token_prob)
    end if
    ys.$append$(next_token)
  end while
  return ys ▷ Matrix with N generated sequences

Each generation with the same source might now produce a variety of different outputs depending on how confident the model is. In the context of seeking varied output, when the same output is repeatedly sampled (as it is generally the most probable), it is referred to as mode collapsing around that output. However, there is no guarantee that this highly probable output is the best one, as it could represent a local maximum rather than the global maximum.

One established sampling algorithm which tries to circumvent the issue of local min/max while also producing many high quality outputs is beam search [50]. The search space is limited by keeping track of a fixed k number of potential candidate sequences at each step and predicting the next output for only that limited selection. The number of candidates k to account for is decided by the *beam size* and as the sequences are built these candidates are dropped and pursued depending on the probability of the next token. In Figure 2.8 one can see how a beam search algorithm could generate a short sentence based on probabilities. The numbers indicate which step the next token (a word in this example) is considered with the filled-out boxes indicating current candidates. The dashed boxes represent tokens that were not pursued.

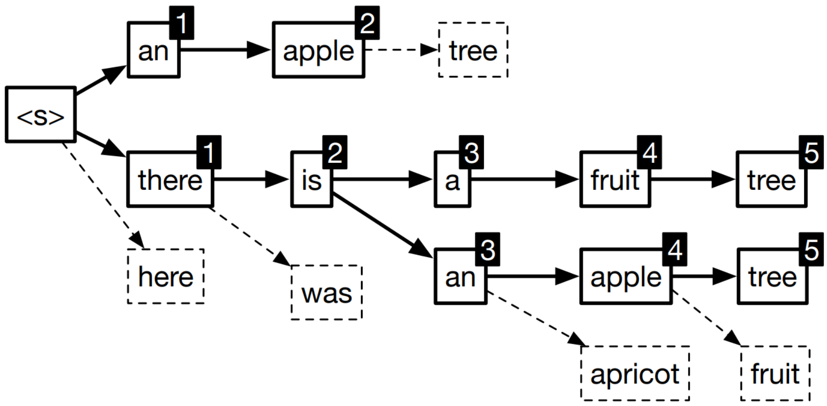

Figure 2.8: Example of beam search from [50], with a start token to the left and *beam size* $k = 2$.

Noteworthy is how at timestep 2, the two candidate sequences "an apple" and "there is" have three potential continuations. Of those, "a" and "an" were the most probable thus the candidate "an apple" was discarded and the other two sequences were continued.

Beam search has its drawbacks however, most notably in the added computational complexity it introduces with producing the output probabilities for all k beams at each time-step. Furthermore, there is no guarantee that the best global solution will be found among the top-k candidates at each step as the risk of pruning it at an early stage is present. The trade-off to minimize this risk is to increase the beam size, subsequently amplifying the existing issue with computational and memory requirements to account for all candidates. The overall quality of the output is substantially higher than that of multinomial sampling but at the cost of extra computation and deterministic results.

3
Dataset

This chapter introduces the dataset used for training, validating and testing the model. The architecture is built for sequence generation and parameters such as the number of layers, types of layers, sampler and optimizer, to name a few, all help shape the model for the intended task.

3.1 ChEMBL

ChEMBL is a public open-source manually curated database of bioactive molecules, predominantly those with drug-inducing properties, maintained by the European Bioinformatics Institute [51]. The used dataset consists of *Simplified molecule-input line-entry system:s* (SMILES, both plural and singular), which is an encoding of the molecular structures of drug-like compounds consisting mostly of carbon and hydrogen. There exists an open-source software toolkit for computational chemistry called RDKit[52] that allows for seamless evaluation and tokenization of both SMILES and other chemical encodings. The subset of SMILES used consists of 500k molecule pairs with RDKit Tanimoto similarity[53] ≥ 0.5.

3.2 Preprocessing

The SMILES might refer to the same underlying molecule twisted or rotated in 3D space, and as such it's important to canonicalize them to avoid duplicates. This is also done with RDKit as an early filtration step before prepearing the SMILES for the transformer. See Figure 3.1 for an example molecule and its canonical SMILE.

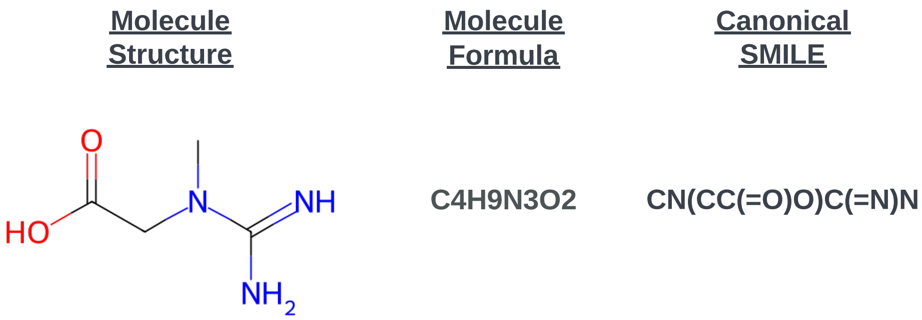

Figure 3.1: Example molecule from the ChEMBL database.

Preprocessing the data for the transformer entails tokenizing the SMILES and grouping all pairs with the same source which allows us to batch pairs of single sources associated with many targets for the new architecture need. If there are cases where more than the desired number of targets exist for a single source, a random non-repeated sample is drawn. If the opposite occurs, targets are drawn with repetition. To avoid outliers, only sources with 10 to 60 targets associated with them are used to avoid excessive repetition of lower cases or major randomness being a factor.

The distribution of targets per source has become interesting with the introduction of k which determines the number of values to sample for each source in one-to-many mapping. To determine a suitable value for k, we analyze the distribution of how many targets map to the same source in the training set. An upper and lower bound is introduced for the number of targets per source that we want to include in the training set. Maintaining a correlation between k and the lower bound is crucial to avoid mode-collapse, as demonstrated by Equation 3.1 where a low lower bound and high k resulted in undesired outcomes.

$$\begin{aligned}
&\text{k} = 10\\
&\text{lower bound} = 3\\
&\text{Source} = [a, b, c]\\
&\text{Source}_{modified} = [a, b, c, a, a, c, b, a, b, c]
\end{aligned} \tag{3.1}$$

As seen in Equation 3.1 the missing 7 targets for $Source_{modified}$ are randomly sampled from the existing targets for $Source$ until it has k number of targets.

The lower bound is inclusive and the decision to choose 10 as a lower bound was found by plotting the distribution of targets per source and picking a value where the majority of sources would be included as seen in Figure 3.2.

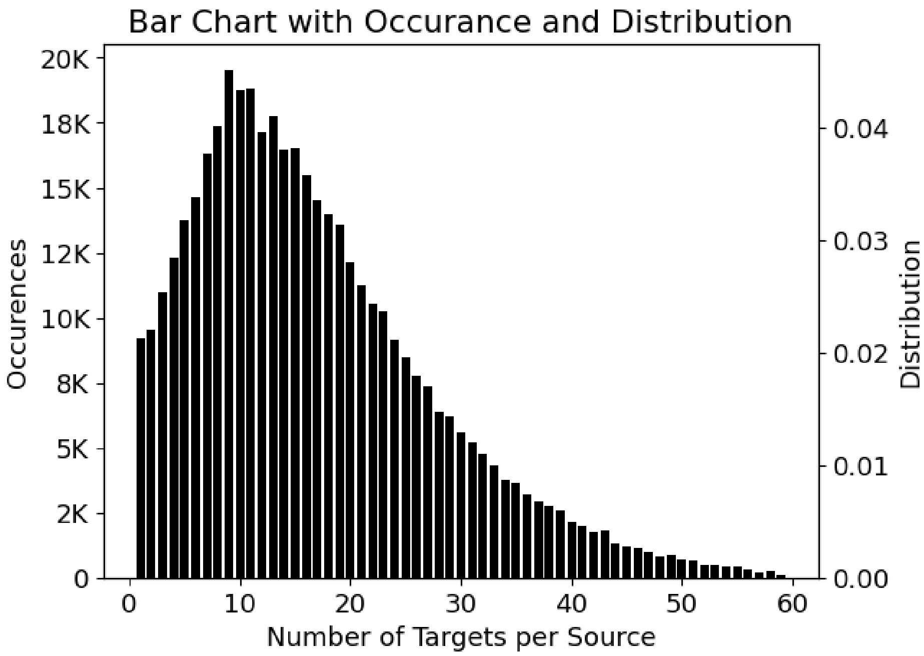

Figure 3.2: Distribution of the number of targets per source

The input is now ready to be fed to the transformer and it is clear what adjustments have to be made, both to accommodate the set of targets but also deliver on the envisioned introduction of variation. While the overall function of the encoder and decoder remains the same in the one-to-many transformer, the decoder now uniquely tracks which associated input sequences it is trying to predict (further explained in Section 4.1). This information is crucial for mapping the correct targets during evaluation and comparing how well the predicted sequences turned out.

4

Methods

This chapter discloses the methods used throughout the project and the motivations for the decisions made.

The base model is a transformer heavily inspired by the work of Vaswani et al.[1] and employs one-to-one (source, target) sequences . This model is the base for our thesis as we are expanding the transformer architecture to support sets in a one-to-many (source, target[]) sequences.

We will introduce the encoder sphere projections strategy in Section 4.1, how it was formed and implemented. Followed is the connected Sinkhorn loss in Subsection 4.1.3 to support the new set focused architecture. How the model is validated in Section 4.2 and later evaluated in Section 4.3 before presenting the hardware used in Section 4.4.

4.1 Encoder Sphere Projections

The premise of this thesis is to expand on the transformer architecture and introduce scalable and efficient architecture-level variation for sequence generation. However, it's simply not a matter of replicating the encoding embedding as they all share the same weights and produce their output deterministically. The inner works of the encoder and decoder is in no need of change, instead the new intensity variation required is added between the two. This transformation projection will be applied to the encoding embedding which then allows for unique sequence generation by batching the variations for the decoder, allowing it to process many sequences independently in parallel.

4.1.1 Constraints

To introduce controlled variations we must first acknowledge some constraints for the transformation of the encoding embedding. This is in no way an exhaustive list of all desired attributes or motivations behind them, but rather a high-level explanation of desired characteristics.

The transformation should therefore:

- be deterministic with respect to the source and how many varied sequences to generate;
- be scalable and efficient when generating more sequences;
- introduce even variation across all sequences;
- ensure independence among the transformations to limit mode collapse.

The first constraint puts the source embedding in focus as the relation connecting it to the target in question should remain true. Expanding the transformation to allow for more generated sequences should not require alteration to the variation already in place for fewer number of generated sequences.

Continuing on accommodation of more sequences gives constraint two, that the overhead introduced should be kept to a minimum by avoiding complicated solutions. Transforming the source embedding aims to result in a linear increase in computation and memory requirement for the actual decoding.

Constraint three focuses on how the transformation should strive to minimize any negative impact on the final output quality. The original model proved to produce satisfying SMILES and this architectural expansion aims to retain that characteristic and thus be applicable to any generic transformer.

Lastly, the evenly spread variation should also be independent of the other transformations. If the source embeddings are assigned similar weighted variations it increases the risk of mode collapse when processed by the decoder. If the transformation is seen as a point in the model dimension space, all transformations should strive to have an equal distance from each other.

4.1.2 Re-usage and Expansion of Positional Encoding

The employed solution presented is a small expansion on the already present positional encoding used in the encoder. Its purpose was to add positional information to the input embedding through a sinusoidal function of the same dimension as the model feature vectors, thus enabling a simple addition of the encoding and the embedding. The function is scalable and efficient with respect to the length of the sequence while introducing controlled variation without any random element. It is therefore deterministic and does not alter the expected transformations when increasing the cardinality. In short, the first two constraints mentioned in the previous section are covered, while aspects related to similar intensity and independence need to be addressed for this expansion to fit the proclaimed needs.

The solution implemented uses the inherent property of orthogonality between the unit vectors in the model dimension space. By computing the Singular Value Decomposition[54] (SVD) of the positional encoding matrix we retrieve a set of independent vectors, all with the same norm and distance from each other. This stems from the definition of SVD as a matrix factorization method that decomposes the given matrix into three matrices.

$$A = U\Sigma V^T \qquad \text{where } A \in \mathbb{R}^{m\times n},\ U \in \mathbb{R}^{m\times m},\ V \in \mathbb{R}^{n\times n} \tag{4.1}$$

such that

$$\Sigma = U^T A V = \mathrm{diag}(\sigma_1, ..., \sigma_p) \in \mathbb{R}^{m\times n} \qquad p = min\{m, n\} \tag{4.2}$$

with diagonal singular values of Σ following the relation $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_p \geq 0$ [54].

U and V are unitary matrices, where if A is a real matrix they are guaranteed to be orthogonal matrices. If $m < n$, the last n columns of V can be ignored as a reduced SVD is instead calculated. These properties are key as the rows of V^T can thus be used as independent positional encoding. They all have uniform norm, retain the same dimensions as the model and share cardinality with the number of sequences we want to generate.

The outcome is a transformation that aims to satisfy the constraints presented in Section 4.1.1 in order to enable the architectural change from a single encoder-decoder relation to a one-to-many spherical projection, illustrated in Figure 4.1. The varied and unique source embeddings are batched and sent to the encoder for continued parallel processing. Instead of the previous pairwise relation, the output embedding now stems from different target sequence embeddings with a shared source sequence.

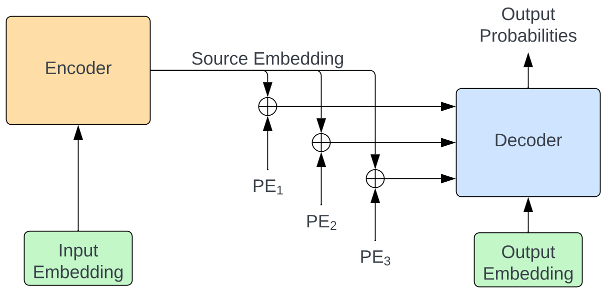

Figure 4.1: Illustration of the high-level architecture with sphere projection on the source embedding.

4.1.3 Sinkhorn Loss

Once the decoder has predicted output sequences for all transformed source embedding the set of outputs is compared with the known targets associated with the source. To encourage variation among the generated output sequences, while striving towards high similarity with the source, we employ pair-wise negative log-likelihood between all predictions and targets. This calculation takes the sum of the negative logarithmic probability for each predicted token in all sequences to generate each of the one-hot-encoded target tokens for their respective sequence, demonstrated in Figure 4.2.

The resulting matrix is summed over each sequence of tokens to result in a cost matrix as described in Section 2.3 that describes the negative log-likelihood of each decoder's predicted output to each of the targets in question. Here we want the model to generate unique targets and as such minimize the total loss of the pair-wise negative log-likelihoods such that each sequence prediction is associated with one unique target. The technique employed is to compute the Sinkhorn distance, as described in Section 2.3, for the cost matrix and thus derive the optimal association of predictions and targets for a minimal loss. The Sinkhorn-Knopp Algorithm employed takes inspiration from [55] with refinement $\epsilon = 0.001$ and 500 iterations of approximation.

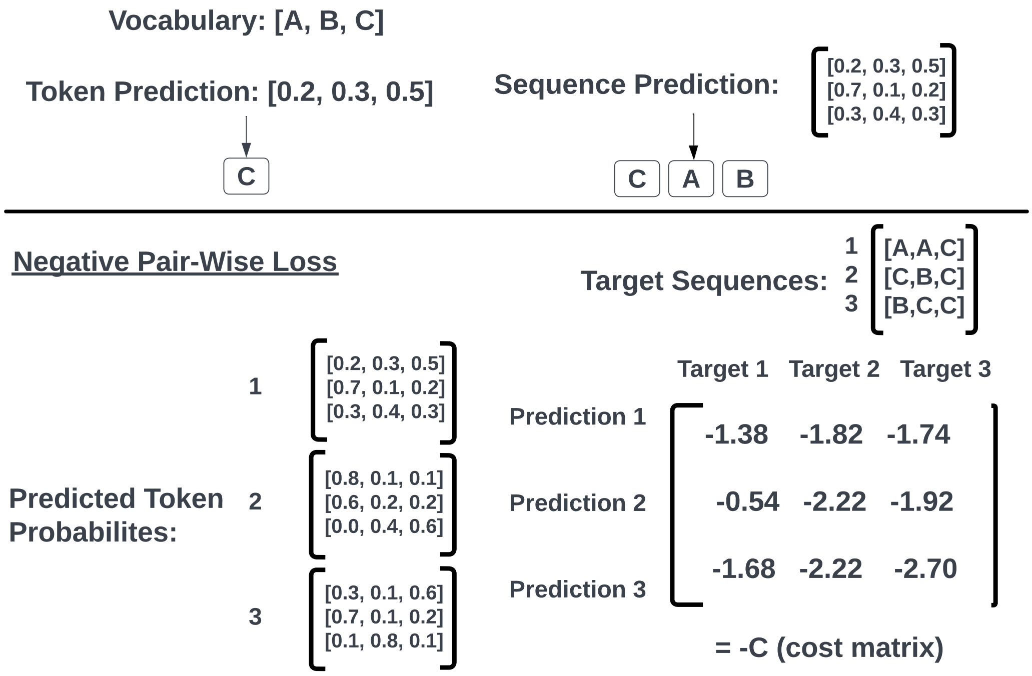

Figure 4.2: Illustration of the cost matrix calculation on three predictions and ground truth targets.

4.2 Validation

Validation asserts that the model trained is continuously improving and thus indicates when it is over-fitting the data or stagnating in advances. When the model is training, the loss derived steers the progress to ensure updates to the model weights result in improved predictions. However, there are two additional important aspects to consider too. The proportion of invalid molecules and the level of mode collapse are valuable metrics to consider as we want the model to generate valid and unique sequences. To give a metric, and thus tangible, notion of the training we introduced *Validation Loss* (VL):

$$VL = \frac{(IM + MC + (1 - PG))}{3} \tag{4.3}$$

where IM is the % invalid molecules, MC is the % of mode collapse and PG is the conversion of the loss to represent the probability for the model generating exactly one of the intended targets.

The PG which is derived from the loss comes straight from the loss evaluation after a full forward pass with a known (source, targets) relation. In contrast, both IM and MC are calculated during greedy inference on the very same source. If the model in question is trained one-to-10, then the inference sampling is also ten sequences per source. The inference is auto-regressive, meaning it generates the next predicted token in the sequence in a greedy fashion from the derived probabilities based on the previous sequence timestep.

4.2.1 Multinomial Validation

This naive VL formula showed promising results and managed to capture the improvement of the teacher forced trained model as it narrowed in on consistently sampling the targets in question. However, in the realm of drug design a favorable generative model must support a much wider exploration of the chemical space. We desire that our model can reach such a requirement and manage to generate different compounds while retaining a high degree of valid molecules of high similarity to the source given. As to better capture these constraints we propose the switch from greedy inference to a multinomial one, further supported in Section 5.1. This introduces a minor change to the established VL as the model loss no longer is applicable with the move from greedy to multinomial probability selection for the predicted tokens.

$$VL_{\text{Multi}} = \frac{(IM + MC)}{2} \tag{4.4}$$

where IM and MC remain the same as in Equation 4.3 but sampled through multinomial selection.

4.3 Inference and Evaluation

Both the base model as well as ours offer the flexibility of supporting beam search and greedy or multinomial sampling at inference. However, considering our objective of achieving a balance between speed and accuracy we are only considering greedy and multinomial sampling since beam search is computationally heavy. This also puts focus on what the architectural changes propagate result wise as the beam search is already a well refined and external method of introducing guaranteed variation. We aim for our model to achieve results that fall between the speed and accuracy trade-offs of multinomial sampling and beam search so as to present it as a reasonable competitor.

4.3.1 Auto-regressive Sequence Generation

While during training the teacher forcing framework is applied to train the model, at inference the model is auto-regressive as no correct target is shared with the model. The source embedding is the main input as the sequence generation initiates and each step after that continues the predicted sequences until an end token is predicted or the max length is reached. This is true for all models evaluated, with the difference being the model architecture and sampling technique used. While beam search follows a procedure explained in Section 2.4, the choice between greedy and multinomial dictates how to decide on the predicted token at each step given the probability output of the model.

4.3.2 Evaluation Metrics

Once an output sequence is generated there are four main points of interest to evaluate it:

- Mode collapse - Repeated output given the same source, only applicable when generating more than one target per source.
- Invalid - The generated sequence is not a valid molecule, i.e it does not correspond to a molecule. To validate that we use the RDKit[52] package.
- Non-Novel - If the output sequence was present in the training or validation set.
- Similarity - How high the similarity to the source SMILE is, calculated with the RDKit[52] package. Converts the SMILE to an encoded molecule, calculates the Morgan Fingerprint[53] and then the Tanimoto Similarity between the source and output. As the train set consists of molecule pairs with similarity ≥ 0.5, such output is referred to as high quality.

The evaluation pipeline is in the same order as above, meaning the degree of mode collapse is the first metric counted. Following is excluding invalid molecules, comparing for novelty and lastly calculating the similarity.

4.4 Hardware and Performance Evaluation

The model supports multiple NVIDIA GPUs for training as it is developed with PyTorch version(1.12.1)[30] and the now standalone package extension Lightning version(1.9.3)[56] to ease the transition to distributed computation.

The hardware used can be seen in Table 4.1. The transformer computation is highly GPU-focused, therefore the remaining system specifications are omitted since the CPU and memory selection did not instigate any bottleneck.

Table 4.1: Available GPUs

Name	Architecture	Memory (GB)	TFLOPs (SP)
K80[57]	Tesla	12	8.73
V100(PCIe/SXM2)[58]	Volta	16/32	14/15.7
A100 SXM[59]	Ampere	40	19.5

In general, the K80s were used for small-scale training, validation and inference, the V100s for training and large inference and lastly the A100s were only used for training. All benchmarking is performed on 32GB V100s if not otherwise specified.

These systems were hosted and maintained in a shared high performance cluster, meaning fully isolated instances for benchmarking were not possible. Allocation of GPUs, CPU cores/threads and memory was possible, but the overall load of the system and disk usage were uncontrolled variables. To mitigate this limitation, we repeat all testing during a similar cluster load and averaged the metrics of interest.

4.4.1 Performance Metrics

The key metric of interest is the execution time of the computation at hand, meaning all preprocessing, model setup and cleanup which are predominantly CPU or disk tasks were excluded. High resource utilization would indicate efficient offloading to, and computation on, the GPU which translates to the end goal of increasing throughput and speedup of execution.

4.4.2 Training

As the model training supports parallel utilization of GPUs the metrics of interest are the degree of speedup, scaling with memory and execution time. PyTorch provides built-in functions to track and log the allocated memory on the CUDA devices used as our goal is to reach a state of computational limitation rather than memory.

Training entails feeding the entire dataset of sources and the desired number of associated targets which defines one epoch. Training a model one-to-one (e.g. the base model) means only one-tenth of the number of targets per epoch is used compared to training a one-to-10 model. This is also an area of interest as it looks at how the new architecture impacts the overall execution time and memory requirements.

4.4.3 Inference

The auto-regressive nature of the inference steered us to only implement single GPU support for sampling. It is however no limitation to simply splitting up the sources over multiple isolated GPUs but as such the parallel speedup will not be compared here. Instead, we focus only on execution time at the optimal batch size given the sampling method at hand and how generating more outputs per source scales with memory and time.

5

Results

In this chapter, the findings and outcomes of the conducted research are presented. The explanation for a few evaluative terms that are used can be found in Section 4.3.2. Throughout the remainder of this thesis, we will continuously refer to our model employing multinomial sampling as one-to-{10, 16, 30} based on the training strategy employed, e.g. the number of targets associated per source. If another sampling method is used it will be explicitly specified.

5.1 Greedy vs Multinomial

As a first step, we have to determine the sampling method to be used in our model. As mentioned in the method we are only considering greedy or multinomial sampling, beam search is excluded from the options due to its high computational demand. Figure 5.1 shows the one-to-10 model for both greedy and multinomial sampling separately denoted as one-to-10 (G) and one-to-10 (M), respectively.

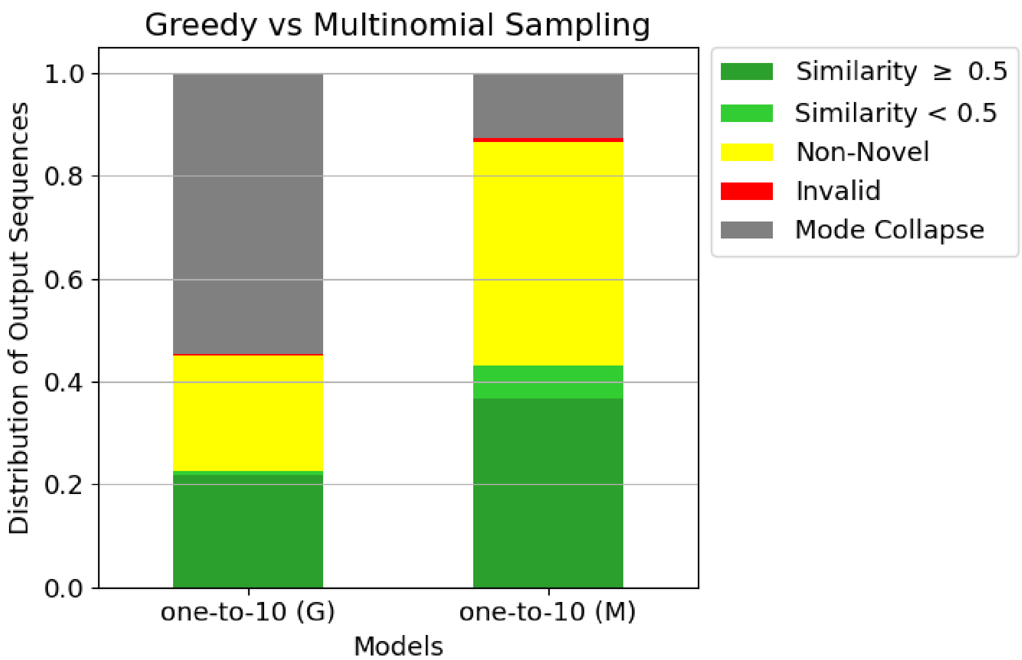

Figure 5.1: Comparison of the one-to-10 model with greedy search (left) and multinomial sampling (right)

Both sampling strategies in Figure 5.1 involved the sampling of 10 targets for 10k sources each. The graph demonstrates that multinomial sampling yields a more diverse output as indicated by the significantly lower mode collapse. Moreover, a higher number of molecules with a similarity score of ≥ 0.5 were identified. Since we want to explore the chemical space, we place value on the diversity that multinomial sampling offers while simultaneously minimizing mode collapse. The one-to-10 model used in Figure 5.1 is trained for 50 Epochs.

5.2 Different one-to-many models

Multiple training strategies were employed that resulted in models trained with varying numbers of targets per source. Specifically, the models trained were the one-to-10, one-to-16 and one-to-30, which correspond to training with 10, 16, and 30 targets per source, respectively. This refers to how many spherical projections were made for each source during training and thus the number of targets to batch accordingly. We then sampled 4 and 16 targets per source for 10k sources as illustrated in Figures 5.2 and 5.3

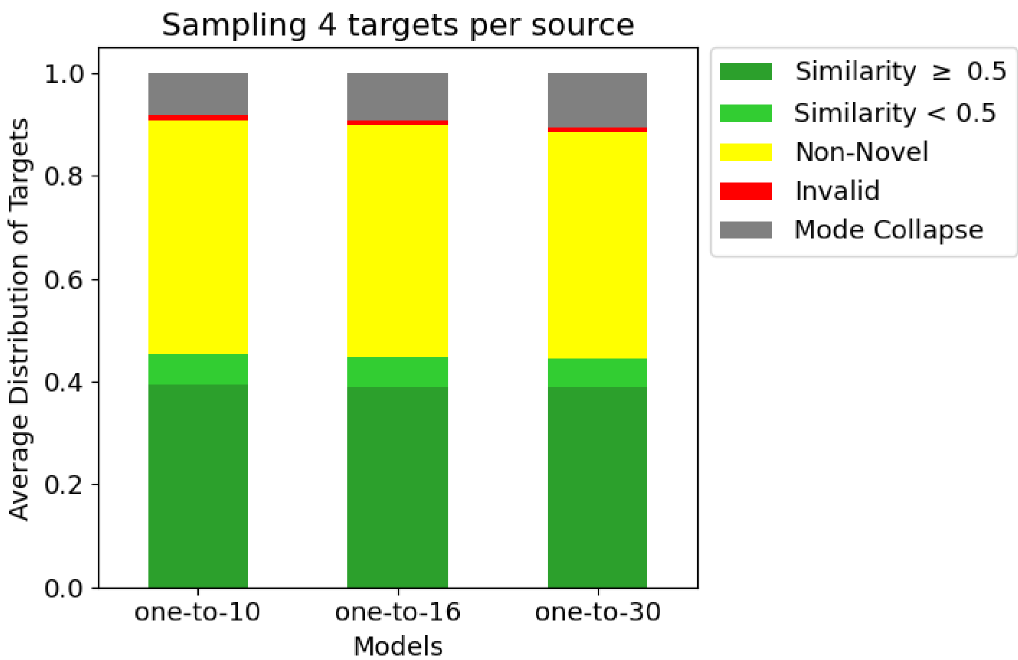

Figure 5.2: Differently trained one-to-many models with the number of targets sampled per source in plot title

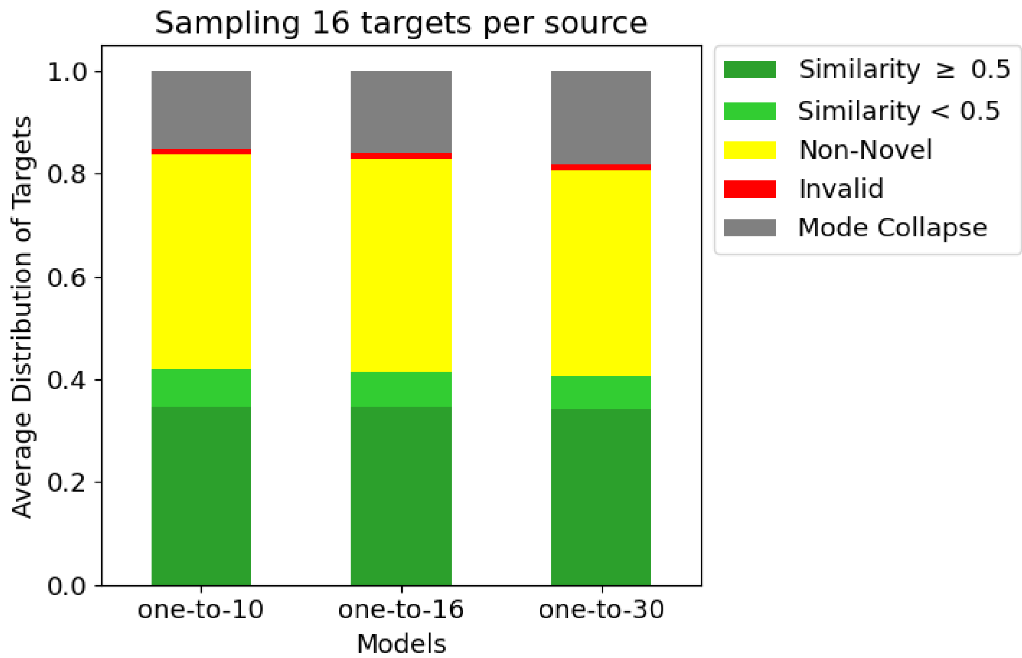

Figure 5.3: Differently trained one-to-many models with the number of targets sampled per source in plot title

Figures 5.2 and 5.3 demonstrate that the performance difference between the models trained for 50 epochs is negligible regardless of the number of targets that are batched together. Notably, the one-to-30 model has been exposed to three times more targets than the one-to-10, however, the impact of this will be shown in Section 5.2.1

5.2.1 Equal Effective Training

When training models with different target batching strategies, the main impact lies in the number of target sequences observed per epoch rather than the architecture itself. Therefore, comparing a one-to-10 model trained for 45 epochs with a one-to-30 model trained for 15 epochs should yield similar results in terms of quality.

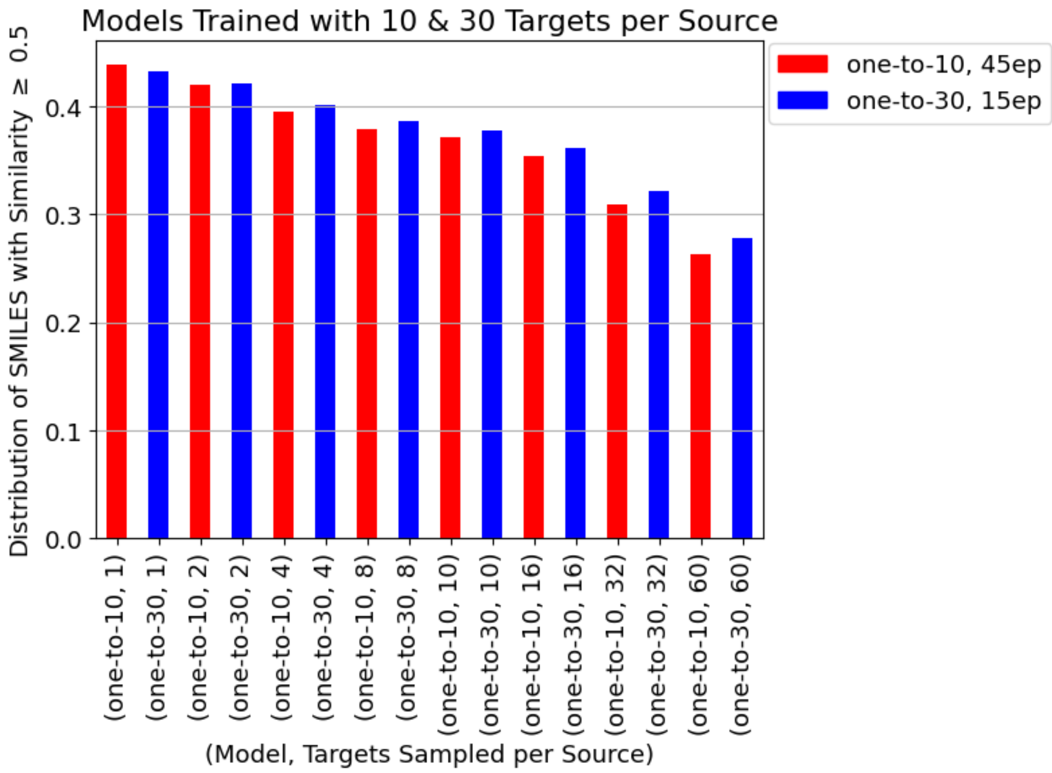

Figure 5.4: The distribution of SMILES with similarity$\geq$ 0.5 at different numbers of targets per source for the one-to-10 and one-to-30 models.

The one-to-30 model holds a slight edge over the one-to-10 model across most sampling sizes as seen in Figure 5.4, especially in the latter half when sampling 10 or more molecules per source. One should however take into consideration the time differences associated with training the two strategies, demonstrated in Subsection 5.6.1.

5.2.2 Output Quality at Higher Inference Size

Leaving the comparison of differently trained models we instead look at how the output quality varies when a model is used to generate more molecules. Training is capped at an upper bound of 60, which means that the model has not been exposed to sources with more than 60 targets. Even so, it is possible to generate more than 60 sequences per source and Figure 5.5 reveals that there is no major drop-off in quality doing so.

The reason for this is that we are after 60 targets adding duplicates in the target batch, as mentioned in Preprocessing 3.2.

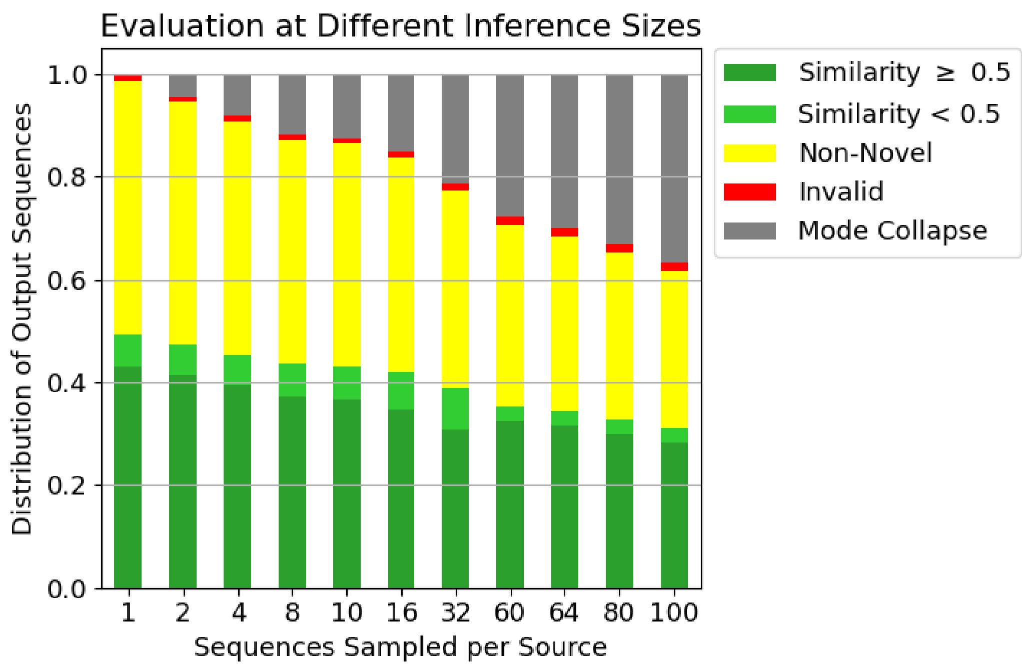

Figure 5.5: Distribution of outputs for a trained one-to-10 model sampling 10k sources at different inference sizes.

At 10 sequences per source we observe 12% mode collapse and 37% high similarity molecules while generating 100 sequences instead demonstrate 36% mode collapse while retaining 28% high similarity molecules. The fraction of sought after molecules does not diminish at the same rate the mode collapse increase.

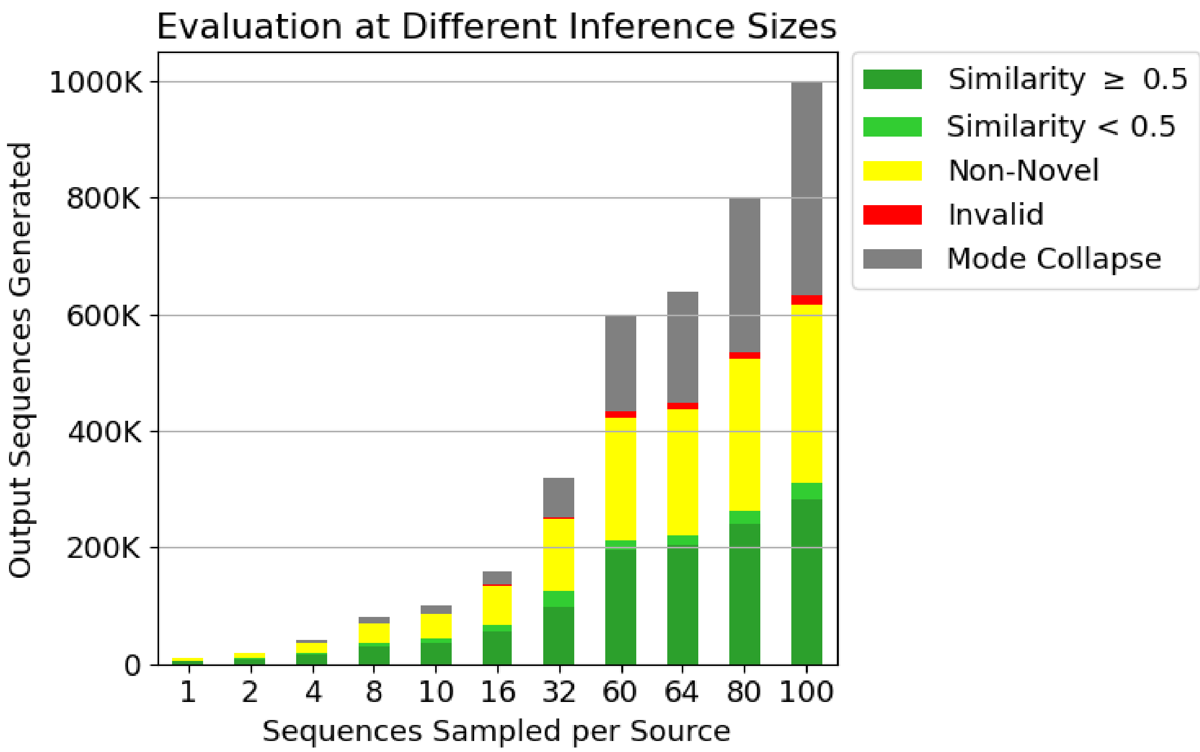

Figure 5.6: Trained one-to-10 model sampling $10k$ sources at different inference sizes.

One important factor to keep in mind as we sample more sequences per source is the total number of molecules generated. As demonstrated in Figure 5.6, the proportion of mode collapse increases but at the same time we observe a larger number of high similarity molecules sampled compared to when we sample 10 targets per source. This becomes particularly valuable when seeking numerous novel variations of the same source.

5.3 One-to-One vs One-to-Many

We now want to compare the new model architecture to the base one-to-one model to see how they both perform at inference. The one-to-10 model and the base model were both trained for 50 epochs with Figure 5.7 showing the respective output given 10k sources.

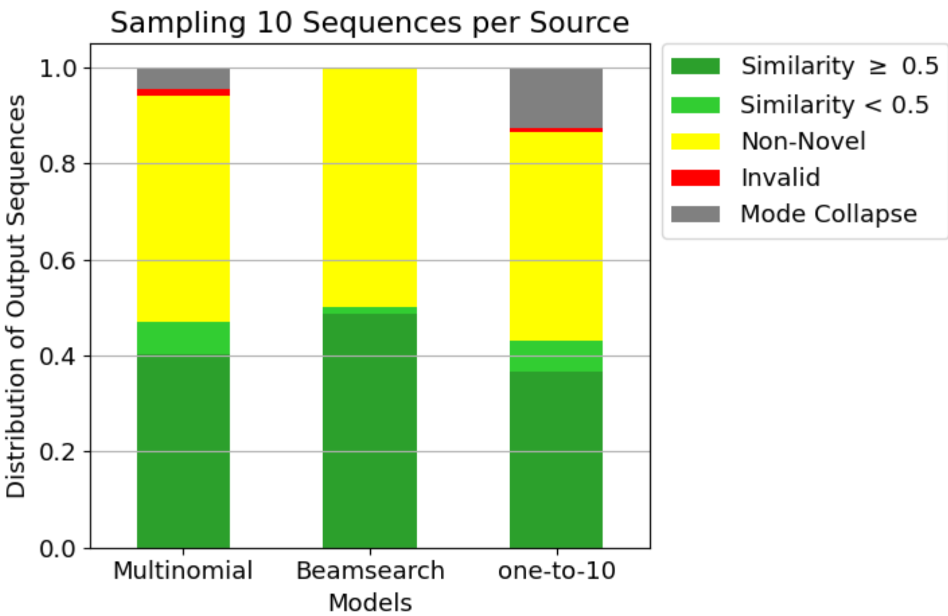

Figure 5.7: Comparing the base model at inference with multinomial sampling and beam search with the new one-to-10 model with multinomial selection.

As Figure 5.7 demonstrates, the new model performs competitively in terms of generating valid molecules but shows a tendency towards mode collapse. While this observation aligns with the expected results of multinomial sampling, continued work on further possible improvements to the new model is explored in the following section.

5.4 Optimal Multinomial Validated Model

As mentioned in Section 4.2.1, it was presumed that a refined metric of the validation, VL_{Multi}, could better capture the desired generative attributes sought after at inference. In Figure 5.8, a one-to-10 model trained for an increasing number of epochs is validated through multinomial inference with 10k sources from the validation set.

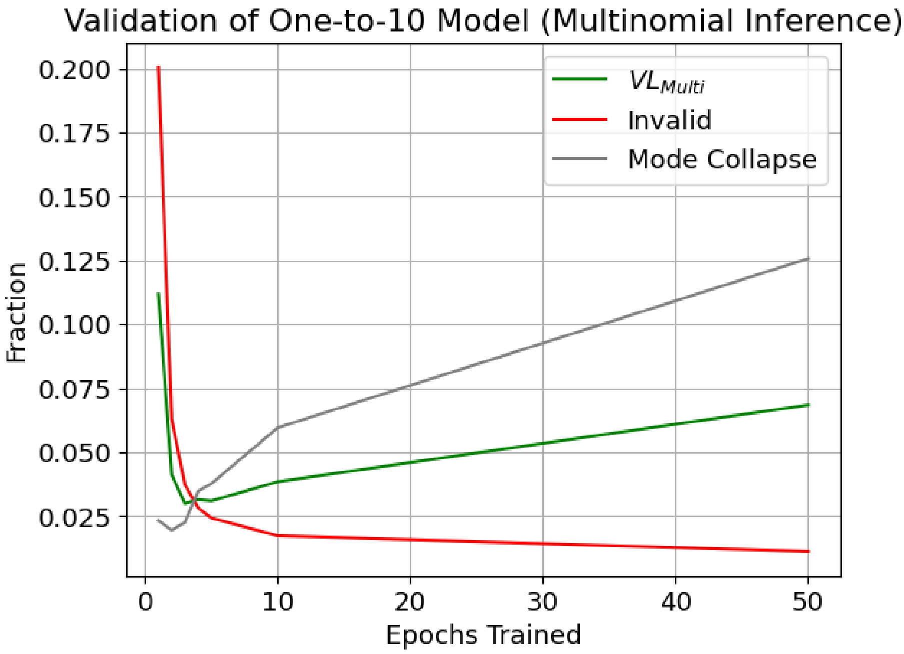

Figure 5.8: Multinomial validation of the one-to-10 model

With the exclusion of the training loss in Figure 5.8, the VL_{Multi} is shown to quite quickly increase after only five epochs of training as the surge of mode collapse outweighs the reduction of invalid samples. The best model in terms of VL_{Multi} is the one trained for three epochs and is henceforth referred to as the *Optimal Multinomial Validated Model*, or simply our optimal model. In Figure 5.9 one can observe this optimal model at inference compared to the original model with beamsearch and multinomial inference and the previously presumed best model, both trained for 50 epochs. Note that the inference is performed on the test set again, not the validation one used to determine the optimal validated model.

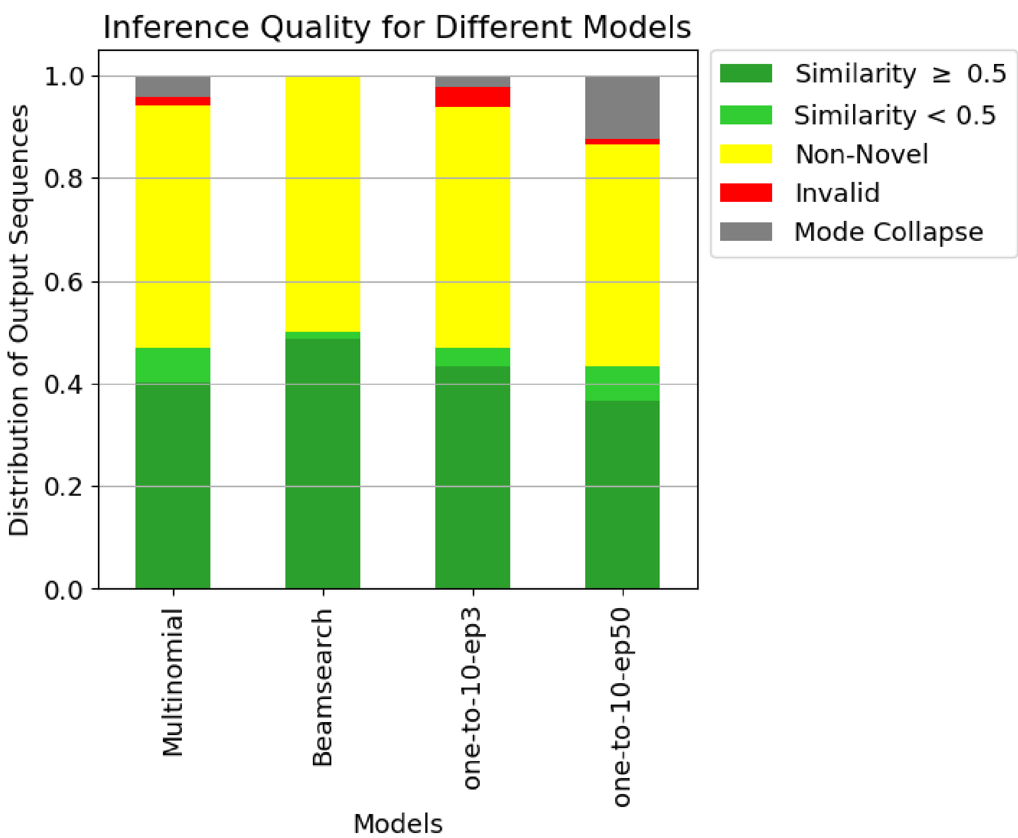

Figure 5.9: Inference at Optimal Validation

The optimal model now outperforms the original one with multinomial sampling in terms of high similarity molecules and thus edges closer to the beamsearch alternative. Looking only at the two new architecture models to the right indicates that the refined validation better captures the desired attribute of generating more valid and high quality samples.

5.4.1 Optimal Validation for other Models and Inference Size

To further investigate the notion of optimal validation a One-to-2 model was trained for 50 epochs with all intermediate steps being validated. Here however, we tried sampling both 2 and 10 targets to see if the validation mostly correlated well with the training strategy applied or at other inferences too.

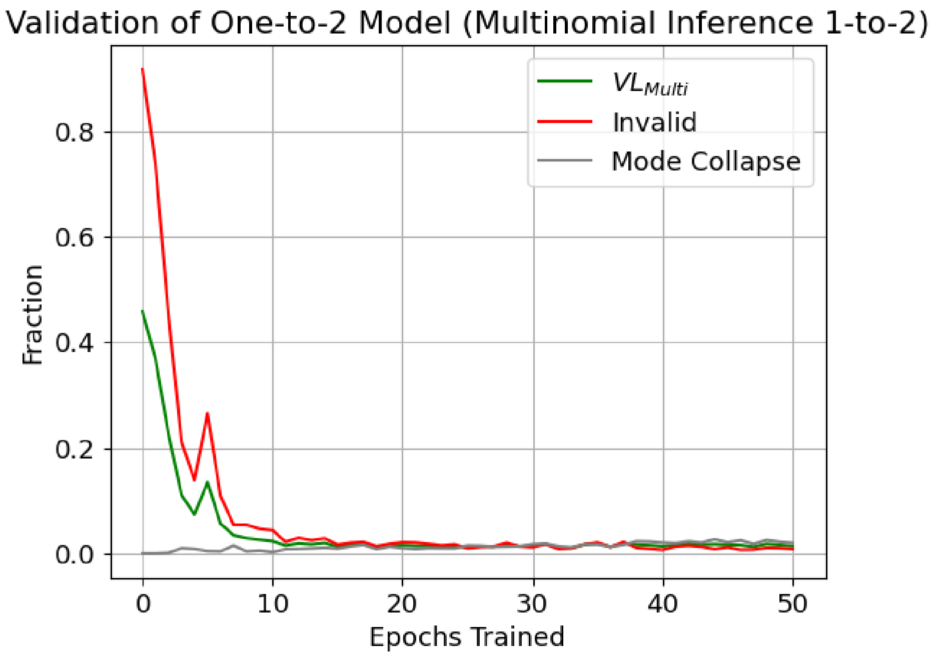

Figure 5.10: The degree of invalid and mode collapse for the trained One-to-2 model.

In Figure 5.10, two samples are generated from each source in the 10k large validation subset. When comparing back to Figure 5.8 and the One-to-10 model, one has to account for the different number of targets observed during training, as the difference between these two models is a factor of 5. While both models improve in VL_{Multi} the first 5 and 25 respective epochs the One-to-10 model shows drastic signs of mode collapse while the One-to-2 model stagnates from epoch 10.

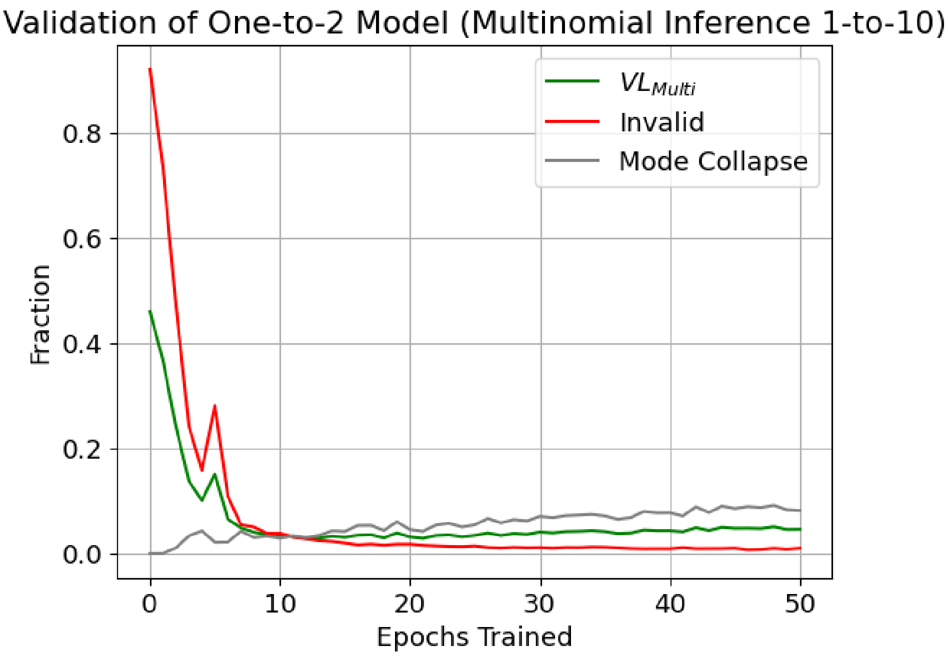

Figure 5.11: Inference at Optimal Validation

Testing at the same inference of 10 samples per source shows in Figure 5.11 that the One-to-2 model has a consistent trend even at the higher inference. The jump in mode collapse is expected but the model is slowly reducing it over time and in this scenario might require more training to mature.

Next, we compared three One-to-2 model epochs of interest, looking at the inference output quality with sampling size 10. The epochs were:

- Epoch 15 - Equates to the best performing One-to-10 model in epoch 3 with regard to the number of targets trained on
- Epoch 25 - Low VL_{Multi} at inference size 2
- Epoch 20 - Low VL_{Multi} at inference size 10

To ease the comparison with the One-to-10 model, epochs dividable with 5 were preferred, albeit the difference was negligible. See Figure 5.12 for the result.

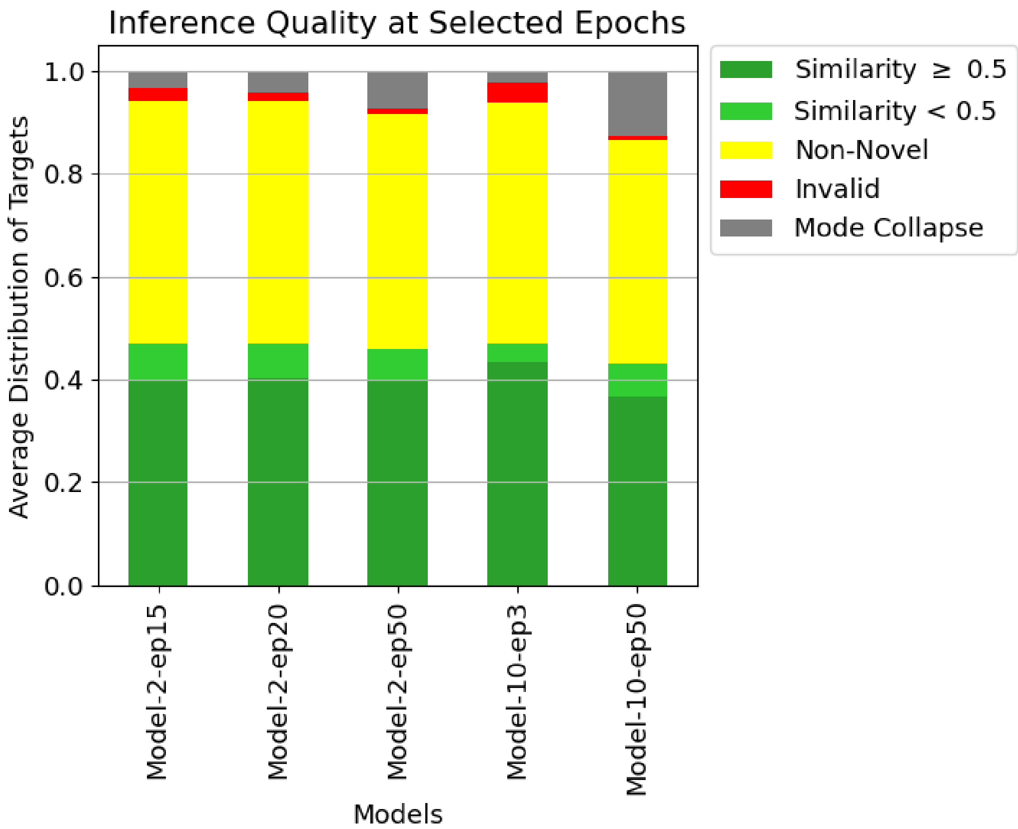

Figure 5.12: Inference at selected training epochs for two models, one-to-2 and one-to-10

The three one-to-2 models (Model-2 in Figure 5.12) all seem to perform within a slim margin. all in between the somewhat better Model-10-ep3 and the equally but worse Model-10-ep50. There is an increase in the mode collapse for both strategies when training for a longer time while the fraction of invalid molecules diminishes.

5.5 Impact of Sinkhorn Approximation Level

With the introduction of pair-wise set comparisons in the loss calculation, it is of interest to explore the extent to which the presumed accuracy correlates with the final inference quality. Our implementation of the Sinkhorn algorithm can be tweaked with two distinct parameters, ϵ for the rate of change and k for the number of iterations. The relation used to ensure adequate total change is $\epsilon \cdot k = 0.5$, with three different configurations tested, see Table 5.1.

Table 5.1: Sinkhorn Algorithm Configurations

Configuration Name	ϵ	k
Base	0.001	500
High	0.0001	5000
Low	0.01	50

As such, three one-to-10 models were trained for 10 epochs each, referred to as Model-High, Model-Low and Model-Base in this subsection. To evaluate them, we compare the multinomial inference quality given the same 10k sources and sampling 10 targets for each source. See Figure 5.13 for the results.

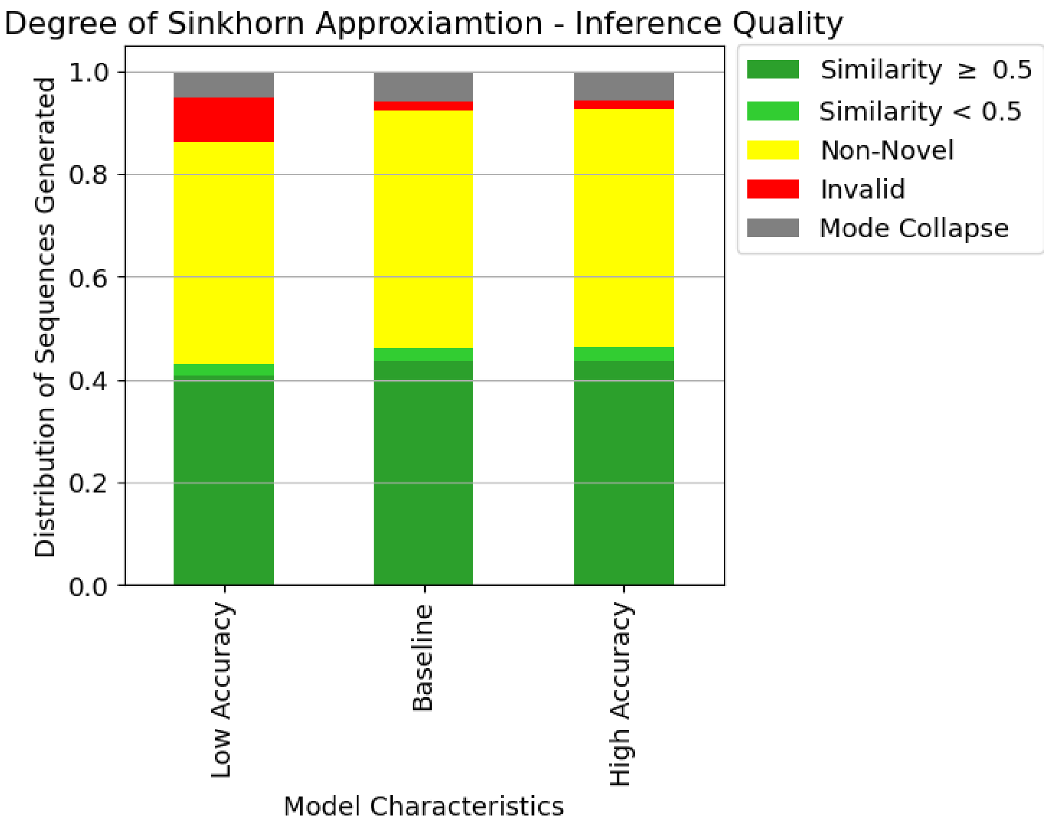

Figure 5.13: Level of Sinkhorn approximation and its impact on the sequences generated.

There is a striking similarity between the output of Model-High and Model-Base while Model-Low is somewhat more prone to generating invalid samples. Noteworthy however is that the training time for these three models are now the same. Compared to Model-Base, Model-High took 4.5x the time to train while Model-Low was faster at 0.65x.

With the output quality known, it is also valuable to see how accurate the approximations really are compared to the ground truth earth mover distance. In Figure 5.14, some configurations of ϵ and iterations k were evaluated with respect to accuracy and time to compute. The tests were done on a single K80 GPU, with the number in each cell representing the average time in seconds to compute the approximation.

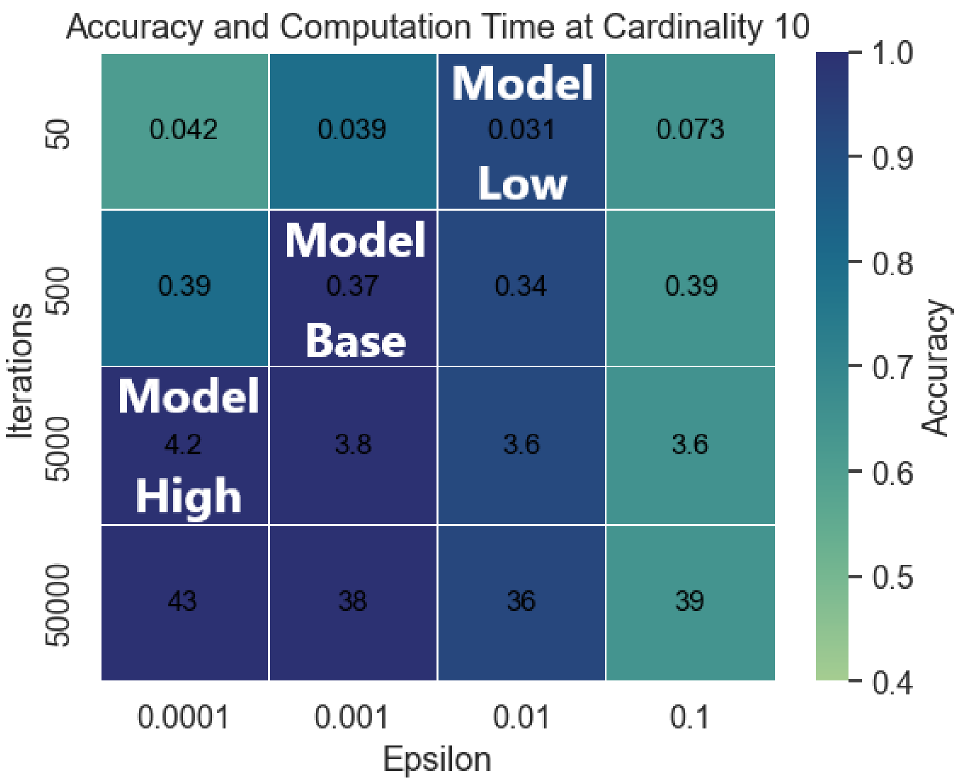

Figure 5.14: Sinkhorn Algorithm, average over 600 batches of points in 3D space. The number in the cells is the execution time in seconds.

The three configurations mentioned in Table 5.1 are marked out in Figure 5.14. It is clear that increasing the number of iterations while keeping the ϵ low produces the most accurate approximations, but the trend seems more tied to the ϵ until it reaches the lowest tested values. The exact accuracy of the tested configurations were 92.3%, 99.7%, and 99.9% for the Low, Base, and High models respectively. The trend in accuracy and execution time stayed consistent when increasing the cardinality to 16 and 32 (see Appendix A.1) which simulates the Sinkhorn approximations performed for the other One-to-16 and One-to-30 models trained.

5.6 Execution Time

An aspect omitted thus far is how changing the transformer architecture impacts the time and resources required to train the model and run it at inference. This section presents the benchmarking data acquired from testing on different hardware setups introduced in Section 4.4.

5.6.1 Model Training

When comparing the training time for differently configured models the other parameter to take into account is the batch size. The data set used for all training benchmarking contained 309k sources, with a separate 18k for validation, which is split up into batches and computed in large matrix formation for efficient computation on the GPU. With more encoder projections, meaning more targets associated with the same source, the batch size had to be reduced in a proportional manner to accommodate the new architecture's memory demands. The largest supported batch size was used to strive towards full GPU computational utilization, but by nature the training was memory bound.

In the following figures, the *Extrapolated Baseline* is introduced to help compare the training of the original model and any new One-to-N model. As the new architecture strategy is comparable to observing more training targets per epoch than the original pair-wise model. We argue that running the One-to-1 model for N epochs results in the same amount of training data observe as one epoch for a One-to-N model. There are other variables at hand related to the preprocessing and sampling of the data, however, those have no implication on the execution time during training.

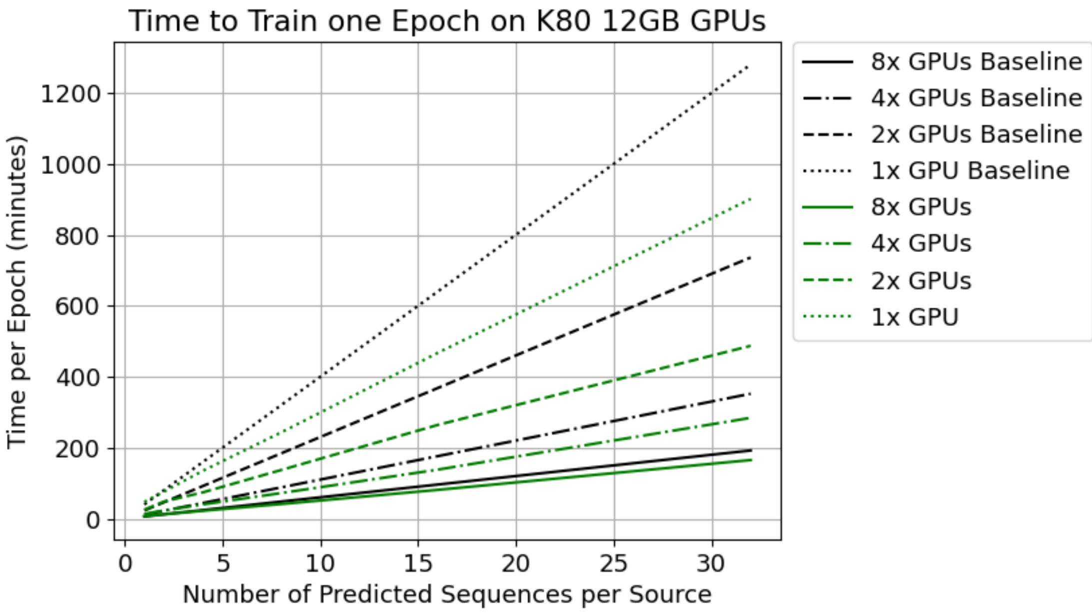

Figure 5.15: Average training time on NVIDIA K80 12GB GPU for varied number of batched targets.

In Figure 5.15 up to 8 K80 GPUs were used to train the baseline model and the new architecture at different batched target set cardinality. Each line type represents a different number of GPUs while the color is for the model used. Looking at the pair-wise relation for any given number of GPUs the new architecture is reliably quicker to train and more noticeable so at a higher target set cardinality. The difference is less noticeable as more GPUs are utilized in parallel however. Both models show a strong linear time increase at all configurations, which is expected for the extrapolated baseline.

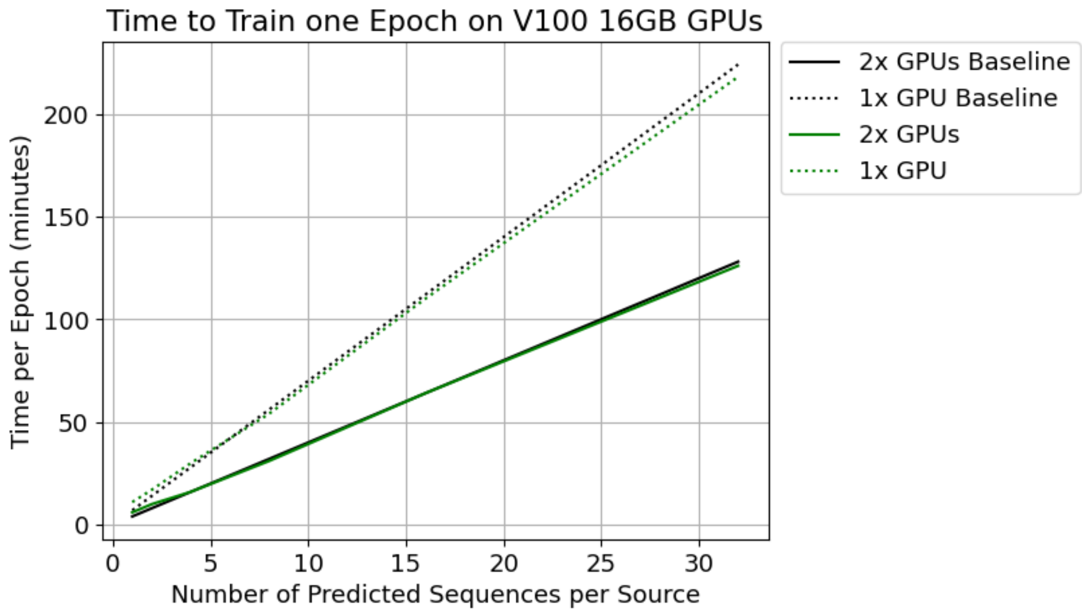

Figure 5.16: Average training time on NVIDIA V100 16GB GPU for a varied number of batched targets.

The next hardware setup tested was the V100 16GB GPUs which Figure 5.16 depicts. The difference is close to negligible and both models scale in a linear manner for the number of predicted sequences per source. The speedup when moving from one to two GPUs is sublinear at an average of 1.72x speedup across all number of batched targets while the baseline achieved 1.75x speedup.

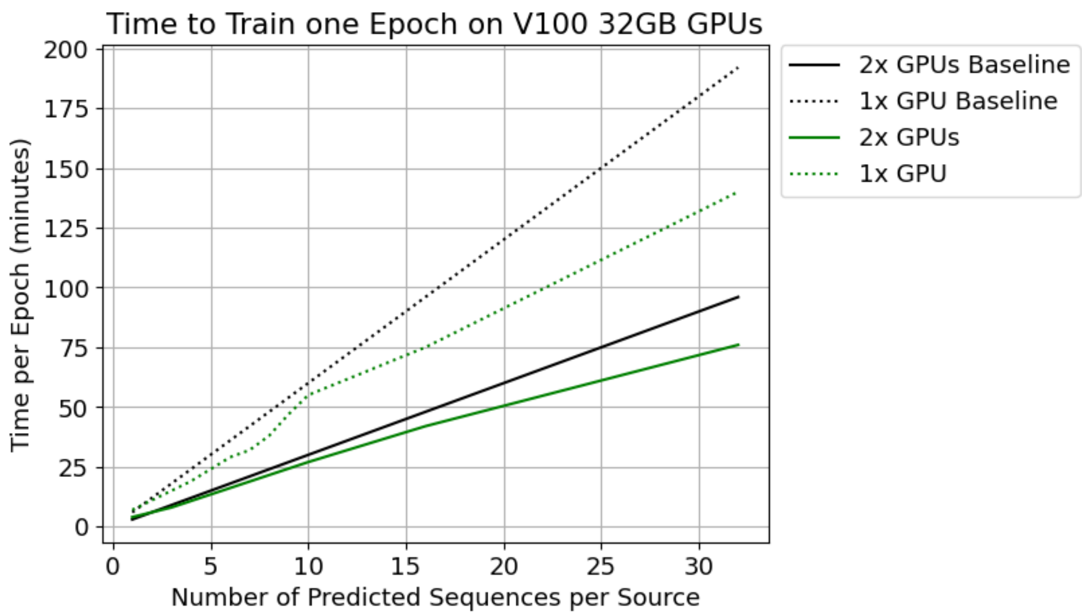

Figure 5.17: Average training time on NVIDIA V100 32GB GPU for varied number of batched targets.

Doubling the available GPU memory to 32GB with the V100 32GB, the new model now outperforms the extrapolated baseline as seen in Figure 5.17. The close-to-linear increase persists for all setups when more predictions per source are made. In the following Figure 5.18, the linear coefficient has been derived through linear regression to further compare the two models on different hardware setups. This looks at the average time it takes to train the new model divided by the number of batched targets used to make it comparable to the original models single one-to-one epochs.

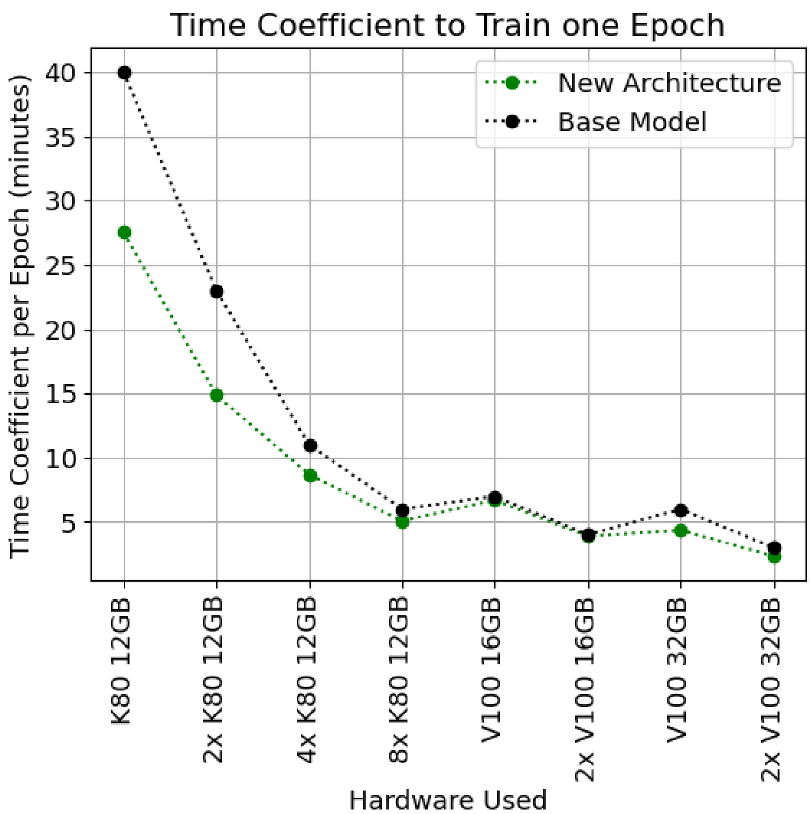

Figure 5.18: Model Training Time Coefficient

Figure 5.18 demonstrate the average increase in time for each hardware configuration tested as more encoder sphere projections are added (with the base model being the average single epoch time). This also depicts the expected speedup when leveraging more GPUs or upgrading to more powerful hardware. For instance, compared with a single K80 the speedup achieved with 2, 4 and 8 GPUs were 1.85, 3.17 and 5.4 for the new architecture. The new architecture shows to scale at least equal to and often better than the original model with the tested hardware.

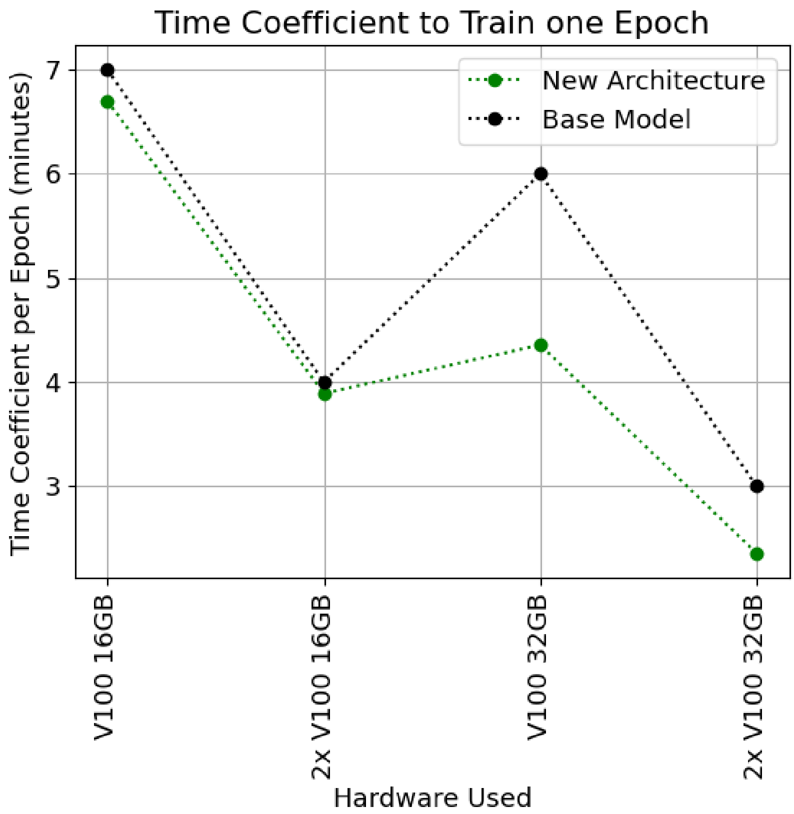

Figure 5.19: Model Training Time Coefficient on V100

Focusing on the V100 GPU in Figure 5.19 gives the unique perspective of increasing the memory of the GPU. The speedup for the new architecture measured 1.54x, comparable to the speedup of instead doubling to two 16Gb V100 at 1.72x. The base model instead saw the memory increase result in a 1.17x speedup while running two 16GB V100 gave a 1.75x boost.

5.6.2 Inference Sampling

In contrast to training, the inference is auto-regressive and was not developed explicitly for multiple GPUs. The performance evaluation was still done on the hardware introduced in Section 4.4, excluding the A100 GPU. Three methods of inference were tested, two in beam search and multinomial sampling on the original model trained for 50 epochs, and one in multinomial sampling on a new one-to-10 trained model for equal many epochs. The inference time is dependent on the length of the generated sequence and thus the quality of the model. The generation ends early when an end token has been detected for a sequence, meaning an untrained model might cut off the sequence too early or reach the length cap unnecessarily. However, no indication of this was seen as both models tested generated output sequences of expected lengths within a few tokens of the given source cardinality.

With the weakest of the GPUs available, the K80, all three models demonstrate a clear linear trend in Figure 5.20. The orange line representing beam search sampling is noticeably slower at higher inference sizes while both multinomial methods for the two models are jointly quicker.

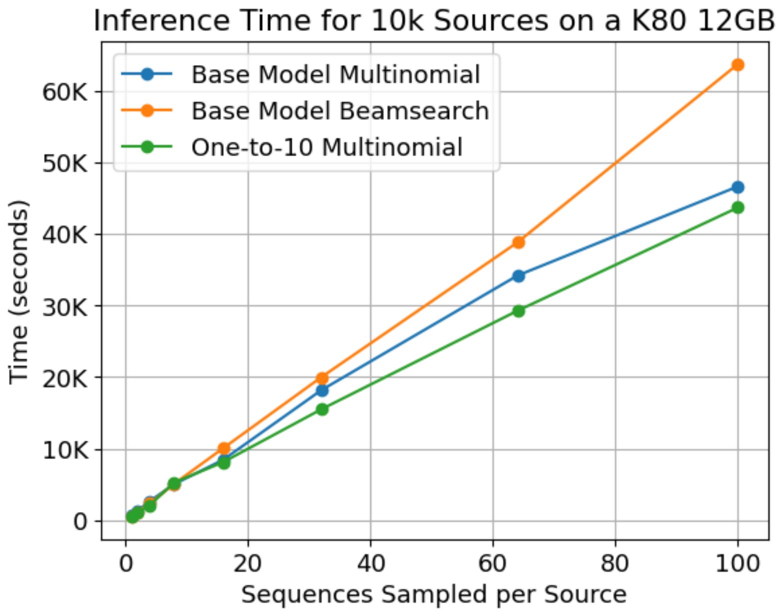

Figure 5.20: Inference time on a K80 GPU at a different number of sequences generated.

Moving to the V100 16GB GPU shows a strikingly similar trend, albeit vastly lower inference times for all tested inferences as seen in Figure 5.21. The one-to-10 model consistently edges out the multinomial base model, averaging a speedup of 1.08.

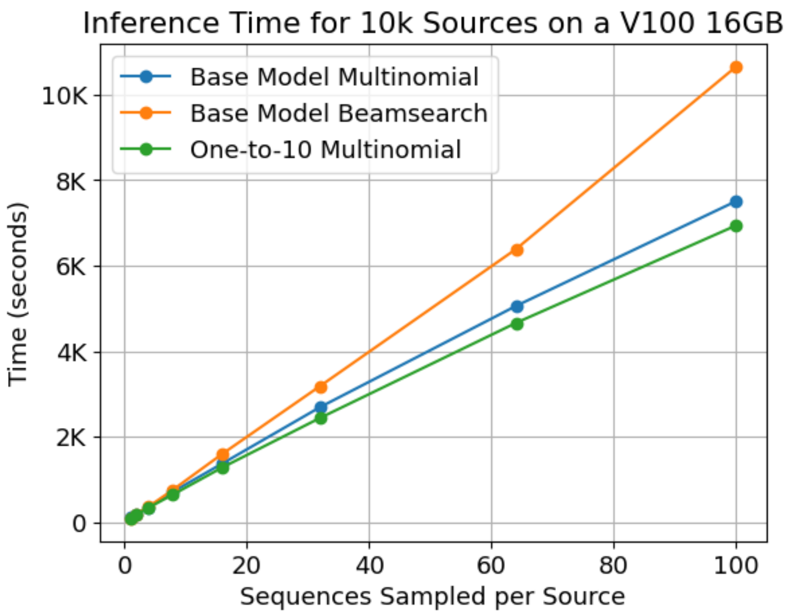

Figure 5.21: Inference time on a V100 16GB GPU at a different number of sequences generated.

Increasing the available memory showed to increase the inference time for all testing. However, the trend and relation between the inference methods remained as demonstrated by Figure 5.22.

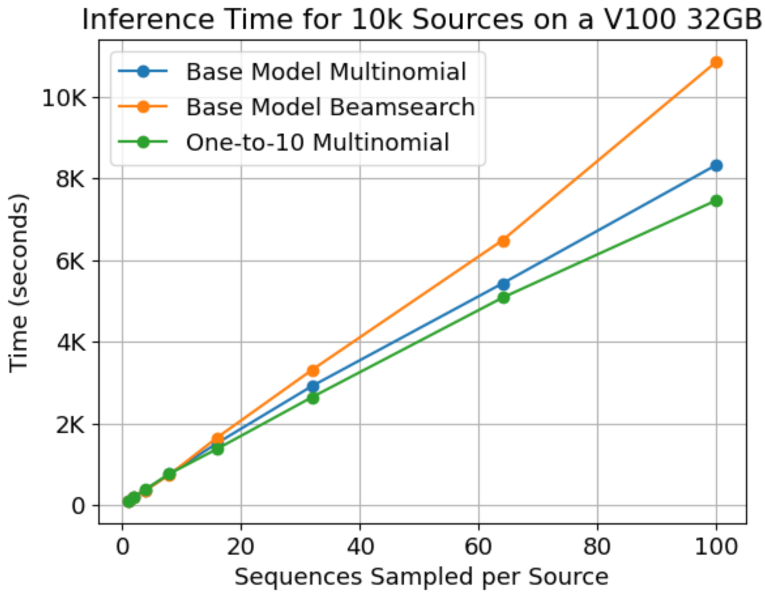

Figure 5.22: Inference time on a V100 32GB GPU at a different number of sequences generated.

Deriving the time coefficient similarly as in Subsection 5.6.1 demonstrates in Figure 5.23 how the new architecture allows for quicker inference than beam search and on par with the multinomial base model.

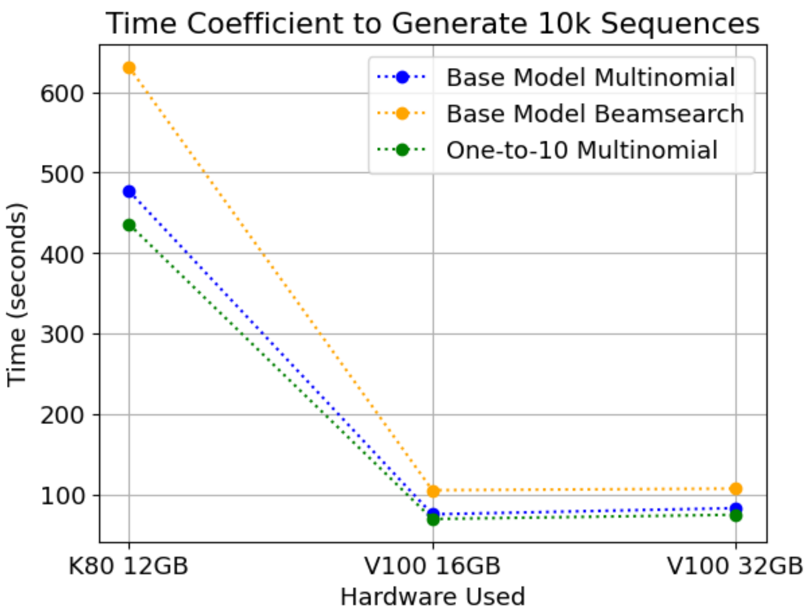

Figure 5.23: Average inference time for the inference methods on different hardware

When comparing the relative speedup, one can expect 40-50% longer runtime for beam search compared to the one-to-10 model regardless of the hardware used. For the base model's multinomial sampling, the same number is closer to 10% across all inference sizes.

6

Discussion and Conclusion

This chapter will present discussions and thoughts concerning the project and draw conclusions based on the results.

The idea of expanding the transformer to handle sets instead of tuples was successful. We had envisioned that our new one-to-many model would surpass the base model employing multinomial sampling in regards to quality but not as good as beam search. Our model achieved comparable performance to the base model running multinomial sampling when trained for the same number of epochs. However, it outperformed it when trained for only 3 epochs, indicating that our model architecture can be further improved.

6.1 Model Evaluation and Training Dataset

The generated results show that the proposed architecture works but it is not perfect. Figure 5.9 shows that the one-to-10 model trained for 3 epochs generates more high similarity molecules while being on par in regards to execution time at inference as Figure 5.21 illustrates.

The model might benefit from a more diverse and extensive training dataset. Increasing the size of the dataset and incorporating various data augmentation techniques can expose the model to a wider range of sequences and enhance its ability to generalize to different inputs. In the used dataset for training, all molecules had similarities ≥ 0.5, it would be interesting to see if the similarities were ≥ 0.7 if that would lead to even more mode collapse. If the goal is to generate more diverse sequences it is worth exploring if our model benefits by adjusting the similarities.

6.2 Impact of Different Model Strategies

During the training the one-to-30 model is more probable to observe all possible targets for a source compared to the one-to-10 model. Specifically when the available targets for the source are 30 or fewer. In contrast, for the sources with 10+ targets, the one-to-10 model has to randomly select between the available targets for that particular source. This random selection introduces an element of chance into the predictions made by the one-to-10 model. To mitigate the impact of this random target selection, the one-to-10 model requires more training epochs. Although the model has been provided with additional epochs, there remains a level of uncertainty that could be contributed to the target selection due to the inherent randomness of the process as illustrated in Figure 5.4.

6.2.1 Architecture's Affect on Execution Time

By expanding the transformer architecture with an encoder sphere projection strategy there are a few key aspects that change how the data is delivered and thus how computation is done. First off, aggregating multiple targets associated with the same source and computing the teacher forcing pattern on all of them in parallel equates to running the original pair-wise training the same number of times in serial instead. The aggregation allows us to skip encoding the same source repeated times which scales as we batch more targets per source, thus increasing the encoding reusage. This however requires new overhead in the Sinkhorn algorithm as a more complicated and excessive loss computation is required, but our results demonstrate that the benefit out-weights it even at low target batch cardinality. Enabling more parallel encoders also indicates an increased leverage of the memory available which is visible when comparing the V100 16GB and 32GB setups. Their respective time coefficient speedups were 1.54 for the new architecture and only 1.17 for the baseline when doubling the memory available.

This all propagates into the demonstrated improvements both during training seen in Figure 5.18 and at inference in Figure 5.23 with regards to the quality of the output. The benefit at inference was seen mostly at higher sampling sizes, leveraging the batched encoder embedding more. In contrast, increasing the GPU memory proved to penalize inference, indicating a computationally bound auto-regression. Given that a very similar output quality was observed, one can argue the benefit of this new strategy is proven.

6.3 Future Work

While the concept of an encoder sphere projection strategy has been proven possible, this section provides a few immediate angles for improvement or further research.

6.3.1 Decoder Strategy

A current limitation is the maximum number of spherical decoder projections supported, being the model dimension. The dimension is derived and selected based on the complexity of the data in relation to what underlying patterns to learn. Forcing more parameters to the model than potentially needed can result in quality issues while also increasing all computation required. Instead, expanding or exchanging the PE function associated with the encoder to enable more embedding transformations is a topic worth exploring. This could allow for more parallel training and inference while also enabling a higher supported inference size.

6.3.2 Beam search and Encoder Variation

The possibility of combining a beam search sampling methodology and the new spherical projection strategy might provide interesting results. Albeit beam search is applied only at inference and as an external sampler, it by nature avoids mode collapse and might unlock more potential from the probability output of the new architecture with the drawback of a longer execution time. Since each varied source embedding produces its own sequence probability, additional work on avoiding mode collapse between the parallel beam searches is expected.

6.3.3 Training

An interesting approach would be to change the training to instead of solely rewarding the model for predicting the correct token we incorporate a diversity-based reward function. Such an approach aims to encourage the model to explore a wider range of valid tokens for a given position, promoting diversity in its generated sequences. This needs to be in relation to the accuracy metric however, as the similarity still needs to be high for the model to continue to generate valid output.

Bibliography

[1] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. arXiv: 1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762.

[2] J. He, E. Nittinger, C. Tyrchan, *et al.*, "Transformer-based molecular optimization beyond matched molecular pairs," *Journal of Cheminformatics*, vol. 14, no. 1, Mar. 2022. DOI: 10.1186/s13321-022-00599-3.

[3] K. He, C. Gan, Z. Li, *et al.*, "Transformers in medical image analysis," *Intelligent Medicine*, vol. 3, no. 1, pp. 59–78, 2023, ISSN: 2667-1026. DOI: https://doi.org/10.1016/j.imed.2022.07.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667102622000717.

[4] Y. Li, S. Rao, J. R. Solares, *et al.*, "Behrt: Transformer for electronic health records," *Scientific Reports*, vol. 10, no. 1, Apr. 2020. DOI: 10.1038/s41598-020-62922-y.

[5] S. He, P. E. Grant, and Y. Ou, *Global-local transformer for brain age estimation*, Sep. 2021. arXiv: 2109.01663 [cs.CV].

[6] C. Wang, Y. Chen, S. Zhang, and Q. Zhang, "Stock market index prediction using deep transformer model," *Expert Systems with Applications*, vol. 208, p. 118 128, 2022, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2022.118128. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417422013100.

[7] Q. Chen, H. Zhao, W. Li, P. Huang, and W. Ou, *Behavior sequence transformer for e-commerce recommendation in alibaba*, May 2019. arXiv: 1905.06874 [cs.IR].

[8] A. Prakash, K. Chitta, and A. Geiger, *Multi-modal fusion transformer for end-to-end autonomous driving*, Apr. 2021. arXiv: 2104.09224 [cs.CV].

[9] L. Dong, S. Xu, and B. Xu, "Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 5884–5888. DOI: 10.1109/ICASSP.2018.8462506.

[10] S. Naveen, M. S. S. Ram Kiran, M. Indupriya, T. Manikanta, and P. Sudeep, "Transformer models for enhancing attngan based text to image generation," *Image and Vision Computing*, vol. 115, p. 104 284, 2021, ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2021.104284. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S026288562100189X.

[11] W. Huang, W. Hu, Y. T. Yeung, and X. Chen, "Conv-Transformer Transducer: Low Latency, Low Frame Rate, Streamable End-to-End Speech Recognition," in *Proc. Interspeech 2020*, 2020, pp. 5001–5005. DOI: 10.21437/Interspeech.2020-2361.

[12] OpenAI, *Gpt-4*. [Online]. Available: https://openai.com/research/gpt-4.

[13] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," 2018.

[14] M. Lachaux, A. Joulin, and G. Lample, "Target conditioning for one-to-many generation," *CoRR*, vol. abs/2009.09758, 2020. arXiv: 2009.09758. [Online]. Available: https://arxiv.org/abs/2009.09758.

[15] J. Li, W. Monroe, and D. Jurafsky, "A simple, fast diverse decoding algorithm for neural generation," *CoRR*, vol. abs/1611.08562, 2016. arXiv: 1611.08562. [Online]. Available: http://arxiv.org/abs/1611.08562.

[16] S. Kumar and W. Byrne, "Minimum Bayes-risk decoding for statistical machine translation," in *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, Boston, Massachusetts, USA: Association for Computational Linguistics, May 2004, pp. 169–176. [Online]. Available: https://aclanthology.org/N04-1022.

[17] S.-H. Han, K. W. Kim, S. Kim, and Y. C. Youn, "Artificial neural network: Understanding the basic concepts without mathematics," *Dementia and Neurocognitive Disorders*, vol. 17, no. 3, p. 83, Sep. 2018. DOI: 10.12779/dnd.2018.17.3.83.

[18] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding: A review," *Neurocomputing*, vol. 187, pp. 27–48, 2016, Recent Developments on Deep Big Vision, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2015.09.116. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231215017634.

[19] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, *Natural language processing (almost) from scratch*, 2011. DOI: 10.48550/ARXIV.1103.0398. [Online]. Available: https://arxiv.org/abs/1103.0398.

[20] J. Chai and A. Li, "Deep learning in natural language processing: A state-of-the-art survey," in *2019 International Conference on Machine Learning and Cybernetics (ICMLC)*, 2019, pp. 1–6. DOI: 10.1109/ICMLC48188.2019.8949185.

[21] B. Van Essen, H. Kim, R. Pearce, K. Boakye, and B. Chen, "Lbann: Livermore big artificial neural network hpc toolkit," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, ser. MLHPC '15, Austin, Texas: Association for Computing Machinery, 2015, ISBN: 9781450340069. DOI: 10.1145/2834892.2834897. [Online]. Available: https://doi.org/10.1145/2834892.2834897.

[22] B. Mehlig, "4.5 restricted boltzmann machine," in *Machine learning with neural networks: An introduction for scientists and Engineers*. Cambridge University Press, 2022, pp. 64–70.

[23] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the Fourteenth International*

Conference on Artificial Intelligence and Statistics, G. Gordon, D. Dunson, and M. Dudík, Eds., ser. Proceedings of Machine Learning Research, vol. 15, Fort Lauderdale, FL, USA: PMLR, Apr. 2011, pp. 215–223. [Online]. Available: `https://proceedings.mlr.press/v15/coates11a.html`.

[24] E. A. Huerta, A. Khan, E. Davis, *et al.*, "Convergence of artificial intelligence and high performance computing on nsf-supported cyberinfrastructure," *Journal of Big Data*, vol. 7, no. 1, 2020. DOI: `10.1186/s40537-020-00361-2`.

[25] Y. Nagasaka, A. Nukada, and S. Matsuoka, "High-performance and memory-saving sparse general matrix-matrix multiplication for nvidia pascal gpu," in *2017 46th International Conference on Parallel Processing (ICPP)*, 2017, pp. 101–110. DOI: `10.1109/ICPP.2017.19`.

[26] V. Karakasis, G. Goumas, and N. Koziris, "Perfomance models for blocked sparse matrix-vector multiplication kernels," in *2009 International Conference on Parallel Processing*, 2009, pp. 356–364. DOI: `10.1109/ICPP.2009.21`.

[27] R. Olivares-Amaya, M. A. Watson, R. G. Edgar, L. Vogt, Y. Shao, and A. Aspuru-Guzik, "Accelerating correlated quantum chemistry calculations using graphical processing units and a mixed precision matrix multiplication library," *Journal of Chemical Theory and Computation*, vol. 6, no. 1, pp. 135–144, 2010, PMID: 26614326. DOI: `10.1021/ct900543q`. eprint: `https://doi.org/10.1021/ct900543q`. [Online]. Available: `https://doi.org/10.1021/ct900543q`.

[28] A. Gupta, G. Karypis, and V. Kumar, "Highly scalable parallel algorithms for sparse matrix factorization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 5, pp. 502–520, May 1997, ISSN: 1045-9219. DOI: `10.1109/71.598277`. [Online]. Available: `https://doi.org/10.1109/71.598277`.

[29] H. Jang, A. Park, and K. Jung, "Neural network implementation using cuda and openmp," in *2008 Digital Image Computing: Techniques and Applications*, 2008, pp. 155–161. DOI: `10.1109/DICTA.2008.82`.

[30] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[31] *What are recurrent neural networks?* [Online]. Available: `https://www.ibm.com/se-en/topics/recurrent-neural-networks`.

[32] G. C. Habek, M. A. Toçolu, and A. Onan, "Bi-directional cnn-rnn architecture with group-wise enhancement and attention mechanisms for cryptocurrency sentiment analysis.," *Applied Artificial Intelligence*, vol. 36, no. 1, pp. 1–25, 2022, ISSN: 08839514. [Online]. Available: `https://search.ebscohost.com/login.aspx?direct=true&db=bsu&AN=160876848&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds`.

[33] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, *Attention-based models for speech recognition*, 2015. arXiv: `1506.07503 [cs.CL]`.

[34] A. Graves, "Generating sequences with recurrent neural networks," Aug. 2013. [Online]. Available: `https://arxiv.org/pdf/1308.0850.pdf`.

[35] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[36] P. Ouyang, S. Yin, and S. Wei, "A fast and power efficient architecture to parallelize lstm based rnn for cognitive intelligence applications," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6. DOI: 10.1145/3061639.3062187.

[37] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014. arXiv: 1409.3215. [Online]. Available: http://arxiv.org/abs/1409.3215.

[38] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. arXiv: 1406.1078. [Online]. Available: http://arxiv.org/abs/1406.1078.

[39] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, 2016. arXiv: 1409.0473 [cs.CL]. [Online]. Available: https://arxiv.org/pdf/1409.0473.pdf.

[40] S. Xu, H. Li, P. Yuan, Y. Wu, X. He, and B. Zhou, "Self-attention guided copy mechanism for abstractive summarization," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online: Association for Computational Linguistics, Jul. 2020, pp. 1355–1362. DOI: 10.18653/v1/2020.acl-main.125. [Online]. Available: https://aclanthology.org/2020.acl-main.125.

[41] S. Raschka, *Understanding and coding the self-attention mechanism of large language models from scratch*, Feb. 2023. [Online]. Available: https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html.

[42] S. Chaudhari, G. Polatkan, R. Ramanath, and V. Mithal, "An attentive survey of attention models," *CoRR*, vol. abs/1904.02874, 2019. arXiv: 1904.02874. [Online]. Available: http://arxiv.org/abs/1904.02874.

[43] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, p. 372. [Online]. Available: http://www.deeplearningbook.org.

[44] A. Lamb, A. Goyal, Y. Zhang, S. Zhang, A. Courville, and Y. Bengio, *Professor forcing: A new algorithm for training recurrent networks*, 2016. arXiv: 1610.09038 [stat.ML].

[45] A. Kazemnejad, "Transformer architecture: The positional encoding," *kazemnejad.com*, 2019. [Online]. Available: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/.

[46] M. Cuturi, *Sinkhorn distances: Lightspeed computation of optimal transportation distances*, 2013. arXiv: 1306.0895 [stat.ML].

[47] S. Shirdhonkar and D. W. Jacobs, "Approximate earth movers distance in linear time," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587662.

[48] R. Sinkhorn and P. Knopp, "Concerning nonnegative matrices and doubly stochastic matrices," *Pacific Journal of Mathematics*, vol. 21, pp. 343–348, May 1967.

[49] G. W. Soules, "The rate of convergence of sinkhorn balancing," *Linear Algebra and its Applications*, vol. 150, pp. 3–40, 1991, ISSN: 0024-3795. DOI: https:

//doi.org/10.1016/0024-3795(91)90157-R. [Online]. Available: https://www.sciencedirect.com/science/article/pii/002437959190157R.

[50] R. Shu and H. Nakayama, "Improving beam search by removing monotonic constraint for neural machine translation," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 339–344. DOI: 10.18653/v1/P18-2054. [Online]. Available: https://aclanthology.org/P18-2054.

[51] D. Mendez, A. Gaulton, A. P. Bento, *et al.*, "ChEMBL: towards direct deposition of bioassay data," *Nucleic Acids Research*, vol. 47, no. D1, pp. D930–D940, Nov. 2018, ISSN: 0305-1048. DOI: 10.1093/nar/gky1075. eprint: https://academic.oup.com/nar/article-pdf/47/D1/D930/27437436/gky1075.pdf. [Online]. Available: https://doi.org/10.1093/nar/gky1075.

[52] G. Landrum, P. Tosco, B. Kelley, *et al.*, *Rdkit/rdkit: 2023_03_1 (q1 2023) release*, version Release_2023_03_1, Apr. 2023. DOI: 10.5281/zenodo.7880616. [Online]. Available: https://doi.org/10.5281/zenodo.7880616.

[53] G. Landrum, *Fingerprints in the rdkit*, 2012. [Online]. Available: https://www.rdkit.org/UGM/2012/Landrum_RDKit_UGM.Fingerprints.Final.pptx.pdf.

[54] G. Golub and C. Van Loan, *Matrix computations*, Third. Baltimor, MD: USA: John Hopkins University Press, 2012, pp. 69–75.

[55] wohlert, *Sinkhorn-solver.ipynb*, https://gist.github.com/wohlert/8589045ab544082560cc5f8915cc90bd, 2019.

[56] W. Falcon, J. Borovec, A. Wälchli, *et al.*, *Pytorchlightning/pytorch-lightning: 1.9.3 release*, version 1.9.3, Feb. 2023. DOI: 10.5281/zenodo.7662458. [Online]. Available: https://doi.org/10.5281/zenodo.7662458.

[57] NVIDIA Coperation, *NVIDIA TESLA K80 - The World's Most Popular GPU*, Oct. 2014. [Online]. Available: https://www.nvidia.com/en-gb/data-center/tesla-k80/.

[58] NVIDIA Coperation, *NVIDIA TESLA V100 - GPU ACCELERATOR*, Mar. 2018. [Online]. Available: https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf.

[59] NVIDIA Coperation, *NVIDIA A100 Tensor Core GPU - Unprecedented Acceleration at Every Scale*, Jun. 2021. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf.

A

Appendix 1

A.1 Additional Evaluation of Sinkhorn Algorithm at other Cardinalities

Additional figures to demonstrate the correlation of cardinality, number of iterations and the tuning rate ϵ has with the accuracy and computation time of the implemented Sinkhorn Algorithm, as introduced in Section 5.5. The number in each cell represents execution time in seconds.

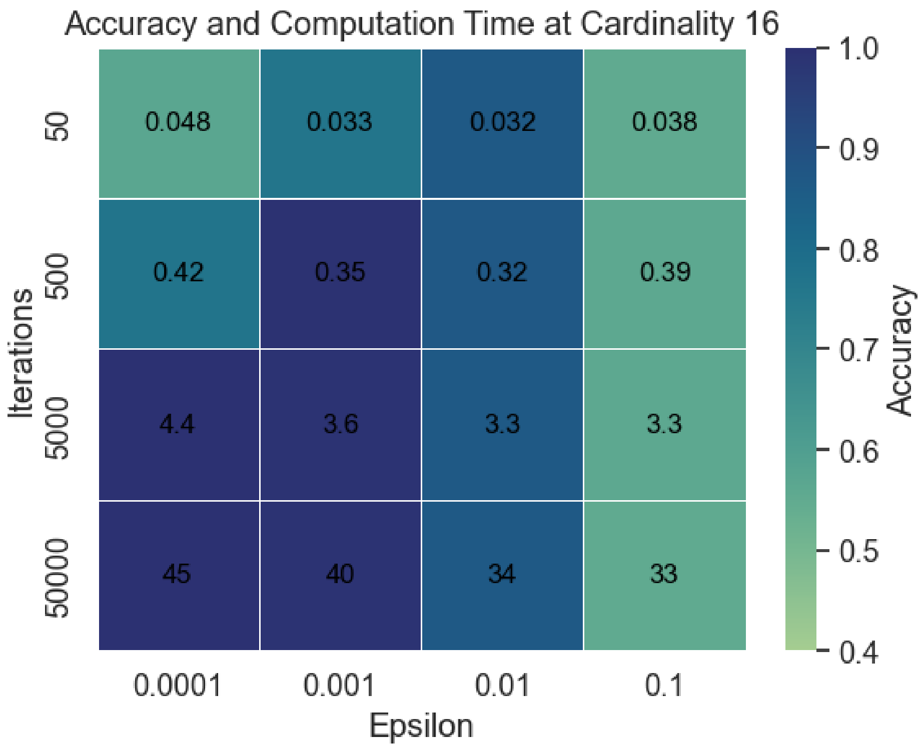

Figure A.1: Evaluation of the Sinkhorn Algorithm averaged over 600 batches of points in 3D space with cardinality 16.

A very similar pattern can be deduced from Figure A.1 as seen in Figure 5.14, albeit the contrast at $\epsilon = 0.1$ is more profound.

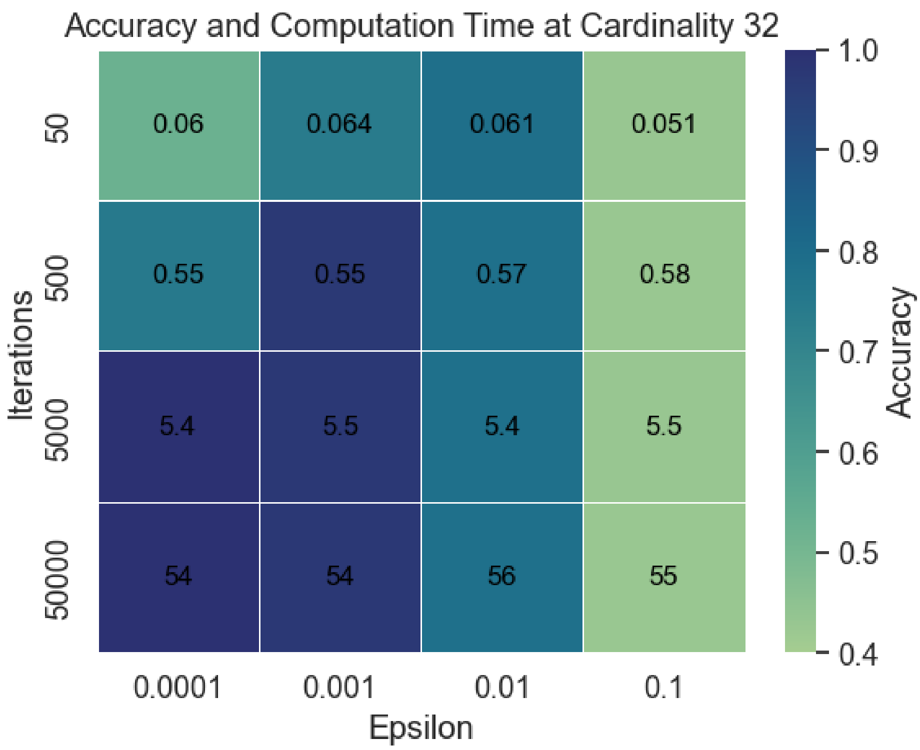

Figure A.2: Evaluation of the Sinkhorn Algorithm averaged over 600 batches of points in 3D space with cardinality 32.

At the higher cardinality of 32, the penalty on the accuracy is magnified when iterating with $\epsilon \leq 0.001$ regardless of the number of iterations. The computational time is mostly tied to the number of iterations, scaling with a factor of 10. It does not however seem to correlate strongly with the cardinality when comparing with the two previous Figures 5.14 and A.1.