Generating SMILES from Mass Spectrometry

Casey Tomlin, Dulce Torres, Esther Mathew, Jesse Maki, Marie Anand

INTRODUCTION

Background

During the discovery process of new molecules, it is common to have created a complex mixture of molecules during synthesis. One of the more tried and true methods of determining the molecules within this mixture is to use Mass Spectrometry (MS or Mass Spec). Mass Spec takes in a sample, ionizes and fragments the molecules, and sends them to a detector. This produces a spectrum of the molecules in the sample, plotted on a graph with the mass-to-charge ratio (m/z) on the x-axis and abundance, or intensity, on the y-axis. Our goal is to implement a machine learning solution to interpreting the raw MS data in SMILES form.

Challenges

Because there are various methods of ionization and mass separation that can influence the results, the first challenge for our project was to source a suitable dataset. For this we wanted to use a basic dataset that used only one type of MS, in our case it was GC-MS. When researching methods, we found a group that had done a similar project, using a CNN and an AutoEncoder to take images of mass spectrometry data and output SMILES strings (Litsa et al, 2023). This group used commercial Tandem MS (MS/MS) data. Due to the higher precision of MS/MS we expected our results to be less accurate.

Additionally, MS spectra are often not unique to the molecule; each molecule can even have several different spectra. To simplify the overall problem - and to more adequately replicate the paper - we trained on single molecules, which is a stepping stone to the more accurate real world scenario where there are multiple molecules in the same spectrum.

METHODS/IMPLEMENTATION

Data Sources

Our dataset comes from the MassBank of North America (MoNA), which is a repository of mass spectral records hosted by the University of California, Davis. It consists of experimental and in-silico libraries as well as user contributions to amalgamate their resources into a more comprehensive dataset. Our dataset is a repository of approximately 18,000 GC-MS spectra with matching SMILES.

To train our SMILES autoencoder, we used a dataset of more than 700,000 unique SMILES from MoleculeNet, a collection of molecular datasets from public databases. These SMILES are unsanitized and canonicalized so there was some preprocessing necessary for input into the autoencoder.

DeepSMILES

One of the approaches we investigated was to convert SMILES to DeepSMILES for tokenization in our models. DeepSMILES is a modified version of the SMILES notation, developed by Noel M. O'Boyle and Andrew Dalke, explicitly designed to enhance usability and application in generative machine learning. DeepSMILES addresses common syntactic issues in SMILES strings, such as enclosed brackets, branching denoted by '(', ')', and rings, which can lead to invalid structures in probabilistic models. DeepSMILES avoids these syntactic issues by using closing parentheses to denote branches and the number of parentheses to indicate the length of these branches, and by employing a single digit for ring closure to specify the size. This approach eliminates the possibility of unmatched symbols. This syntax allows for easier processing by deep neural networks and ensures that the strings are more likely to represent valid chemical structures without losing chemical information.

AutoEncoder

First, we worked to implement an AutoEncoder. The SMILES autoencoder (AE) takes in a set of randomized shuffled SMILES strings, encodes it to a SMILES embedding, and decodes the embedding to construct the sanitized and canonicalized SMILES string through a bidirectional gated recurrent unit (GRU) architecture. The GRU was chosen because it implements a solution to the vanishing gradient problem in recurrent neural networks where backpropagation causes the gradient to become smaller until it does not update weights significantly, thereby increasing error. Vanishing gradients make it more difficult for the model to memorize characters farther away in the sequence. The latent dimension should be representative of a completely randomized SMILES string which the AE will decode to a sanitized and canonicalized SMILES. The Mass Spectra transformer architecture will read in the MoNA dataset and create a latent representation of the spectra with the same dimensionality as the encoder output from the AE.

Prior to training the AE, the sanitized SMILES strings are read in from a csv and a sequence length column is generated which is important for future processing. This is because we limit the length of the SMILES strings inputted into the AE to remove outliers. The maximum SMILES sequence length was 77, and we padded shorter strings with whitespace to have a length of 77. We also filter SMILES inputs to exclusively include the following characters: C, =, (,), O, 1, 2, N. These are the eight most frequently occurring characters in SMILES strings as seen in **Figure 1**. This allows us to train the AE to capture the qualities of the majority of SMILES strings and avoid incorrect handling of characters for which we do not have enough training data. Each sanitized SMILES string was then randomly shuffled to create a randomized string. The randomized string would be the AE input and the sanitized string would be the target output.

After filtering and creating the sanitized and randomized SMILES character strings, we converted each character string into a list of integer classes with a dictionary mapping. We then

applied one hot encoding on the integer lists so that our loss metrics were not artificially impacted by the magnitude of the integer. This one hot encoded data was converted into our randomized input and sanitized target output PyTorch tensors. We created a training dataset with 500 SMILES sequence which was also limited by the kernel.

When implementing the AE, we ran into the problem of the model continuously predicting the same class (often carbons) repeatedly for every sequence position. We troubleshooted this by increasing the hidden dimensions from 10 to 512. A hidden dimension size of 512 was used for the SMILES autoencoder in a previous study to create an end-to-end deep learning framework for translating mass spectra to de-novo molecules (Litsa et al, 2023). Increasing hidden dimensionality allowed the model to embed more information about the input sequence, thereby generating more variance and relevance in outputs. Switching from Tanh activation to SoftMax activation also improved the model because we are selecting the class with highest probability to create the predicted sequences.

The final architecture of the AE includes a bidirectional GRU encoder and decoder with three layers, hidden layer dimension of 512, sequence length of 77 (length of SMILES strings), input size of 9 (number of unique characters including whitespace), learning rate of 0.0001, and activation function of SoftMax. The autoencoder was implemented with PyTorch.

Transformer

We also evaluated a Transformer model as a potential framework for predicting SMILES output from GC-MS spectra.

From our raw SMILES data, we removed all information about stereochemistry, as Litsa, et al noted difficulty when including this information. We removed missing data and ensured validity of the SMILES strings as well.

To begin preparing the data for our model, we initially examined our input data. In the output SMILES data, we visualized 45 unique characters, shown in **Figure 1** below.

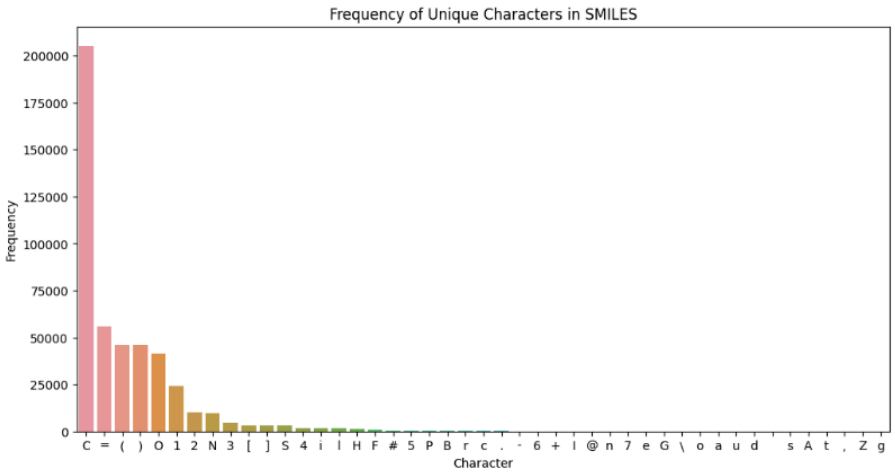

Figure 1: *Visualization of the frequency distribution of unique characters in our SMILES training dataset.*

Similarly, we examined the GC-MS input data, and found 517,627 unique tuples. The task at hand was to convert from a vocabulary of 45 characters to that of 517,627.

To start, we filtered our SMILES string to fewer than 77 characters to remove outliers, and filtered out the paired GC-MS spectra, as discussed in previous methods. We then converted the SMILES strings to DeepSMILES, which were tokenized by a custom dictionary of characters based on our dataset. We attempted two methods of tokenization: by unique character, and by four-character substrings.

Using unique character tokenization, our DeepSMILES vocabulary size decreased to 36. DeepSMILES sequences were filtered so that all were between 25-30 characters, giving us a fairly uniform maximum length and roughly 1500 samples for training. Though we would have liked to use the entire >18,000 datapoint matched set for training, this was not computationally feasible.

The DeepSMILES input was then one-hot encoded to prevent bias in training.

We set a max length for both the DeepSMILES and GC-MS data, so each set had uniform length. DeepSMILES had a max length of 29 characters, and GC-MS data had a max length of 300. Spectral data was normalized to avoid bias from initial input. These data were broken into training and validation sets on the same indices and loaded into DataLoaders with a batch size of 16 for processing. We maintained a low batch size to prevent memory overload.

The additional method of tokenization consisted of breaking up the MS strings into substrings consisting of unique mass to charge ratios. This resulted in a vocab size of 71,353. The longest MS string was observed at a length of 522, thus shorter strings were padded to a length of 522. Each substring was converted into a corresponding integer through mapping and included in the input tensor. Embeddings were generated through the builtin one hot encoded tensor function provided by PyTorch. Since the resulting embeddings were too computationally demanding on both local and Kaggle GPU resources, average pooling was used to reduce the size of the embeddings. The same process of data loading and batch size parameters used in the aforementioned tokenization method were applied.

Transformer Architecture

To implement our Transformer, we used the PyTorch package. We based our code on the flagship Transformer paper architecture and edited to work with the specifics of our work.

In the Transformer class, we defined the number of tokens, the dimensions of the model, the number of heads, the encoder layers, the decoder layers, and the dropout percentage for regularization. Our inputs were embedded in separate embedding steps due to the different shapes: the GC-MS data had two features: both mass to charge ratio and intensity, while the DeepSMILES data did not have a second feature. Positional encoding was applied, which ensures that the transformer maintains information about a token's position in a sequence. Each input had separate positional encoding steps due to the different maximum sequence lengths.

We applied a target mask so the transformer can only attend to the previous position; future tokens are masked away. Padding masks were used for both the source and target data to ensure that the transformer was not fitting to our padding tokens. Custom padding tokens were used for the DeepSMILES data ('<PAD>'), while our GC-MS data was padded with '0'.

We used CrossEntropy loss, as this complicated problem boils down to classification of input MS data to output DeepSMILES tokens. For optimization, we used the Adam optimizer with an initial learning rate of 0.001. We trained for 120 epochs.

Transformer + AutoEncoder

The integration of the AE bidirectional GRU decoder with a transformer encoder was explored as a potential framework to generate SMILES from mass spectrometry data. This method was pursued as a way to mitigate syntax errors that arise when generating SMILES strings, especially when working with limited data. Integration and development of such a model was three-fold, with the first step being the aforementioned training of the AE model. The encoder output was then extracted, serving as the target output of the spectra encoder. Input to the spectra encoder consisted of prepared MS data. Appropriate MS data was selected based on corresponding SMILES length. MS data was filtered out if the corresponding SMILES sequence exceeded a length of 77. The MS strings were then tokenized into a substring of characters, where each character was associated with an observed mass to charge ratio. Padding was added to each string sequence that did not reach a minimum length of 524. Each token was then mapped to an integer value, and embeddings were generated through a one hot encoded tensor.

The resulting embeddings were a tensor with a dimension of sample size, 524, 506884. Average pooling was utilized to transform the spectra embeddings to ensure shape compatibility. This resulted in an embedded tensor with the same dimensionality as the output from the GRU encoder. Since the primary goal of the transformer encoder was to essentially output embeddings that closely resembled the GRU encoder output, an RMSE loss metric was utilized during training. Consistent decrease in loss was observed during training. Additional work in the integration would consist of combining the transformer encoder with the GRU decoder.

However, it is believed the integrated model performance could be compromised due to the MS embedding preparation where large amounts of data were compromised from average pooling.

Hybrid Encoder-Decoder Neural Network for Alternative Molecular Identification

The Hybrid Encoder-Decoder Neural Network is designed to address the challenges of generating SMILES strings directly from mass spectrometry (MS). Instead of employing sequence classification or generative modeling approaches, this model focuses on mapping spectral data to latent representations aligned with chemically meaningful substructures. This is made possible by leveraging SMILES Pair Encoding (SPE) pre-trained vectorizer, which tokenizes SMILES string into high-frequency substructures, simplifying the prediction process and reducing syntactic noise. This integration ensures that the model learns compact and

chemically interpretable representations, bridging the gap between complex spectral inputs and valid molecular outputs.

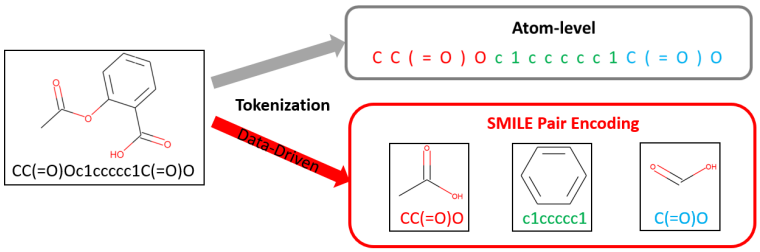

Figure 2: *SmilesPE Substructure Tokenization* https://github.com/XinhaoLi74/SmilesPE/tree/master?tab=readme-ov-file

The preprocessing pipeline uses SMILES Pair Encoding (SPE) for molecular structure tokenization. This approach leverages Word2VEC embedding with a specified parameter: 100-dimensional vector space, a context window of 5, and a minimum count of 1. The implementation also includes the SPE2VEC class, adapted explicitly for Gensim 4.0.0. This tokenization process transforms variable-length SMILES into a standardized 100-dimensional vector representation to create a consistent input format for the decoder network.

Mass spectrometry data (mass-to-charge:intensity pairs) transforms into a 2D matrix representation through a binning process. The preprocessing employs a 200-bit resolution for m/z values and 100 bins for intensity, creating a standardized grid. A noise threshold of 0.01 is applied to filter out low-intensity signals, and intensities are normalized to the relative max peak. These steps were necessary in order to pass through the 2D convolution neural networks.

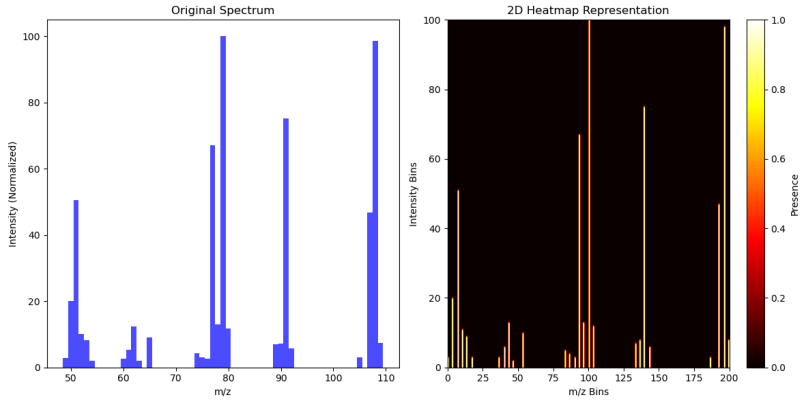

Figure 3: *Example image of post-processed spectrum*

The dataset was split into training, validation, and testing sets with ratios 70:10:20. The data pipeline implements TensorFlow Dataset API for efficient batch processing with a batch size 32.

The encoder components implement a hybrid architecture combining convolutional and recurrent neural networks. The input layer accepts the 2D spectral matrices (200x100x1) and processes them through two convolution blocks. Each block consists of a Conv2D layer with ReLu activation, batch normalization, max pooling, and a dropout rate of 0.3. The resulting features are then processed through a bidirectional LSTM layer, creating context-aware encoding of the spectral features. In addition, L2 regularization (1e-4) is applied throughout to prevent overfitting.

The decoder network transforms the latent representations into molecular vector predictions through dense layers. Starting with the encoded features, it processes them through two fully connected layers of 256 and 128 units, respectively, followed by batch normalization and dropout. The final layer outputs a 100-dimensional vector corresponding to the learned molecular representation. This architecture balances complexity with computational efficiency while maintaining the ability to capture subtle molecular features.

The model employs an adaptive training strategy with the Adam optimizer initialized at a learning rate 1e-4. The loss function uses mean squared error, which is appropriate for the regression nature of the vector prediction task. Training includes early stopping with the patience of 10 epochs and learning rate reduction on plateau, decreasing by a factor of 0.5 after five epochs without improvement. Mean absolute error serves as an additional metric for model evaluation.

The practical utility of this hybrid architecture extends significantly into unknown compound identification through a streamlined prediction pipeline. When presented with an unknown mass spectrum, the encoder component processes the spectral data through its convolutional and LSTM layers to generate a latent representation in the same vector space as the SPE-encoded molecular structures. This encoded vector can then be compared against a database of known SPE-vectorized molecules using similarity metrics such as cosine similarity or Euclidean distance. By ranking these similarity scores, the system can propose candidate molecular structures that likely match the unknown spectrum, effectively creating a ranked list of potential molecular matches.

Results show the model's potential to narrow down compound candidates and resolve underlying substructural features from MS data. Below are images from the best fit and worst fit vector predictions and training history.

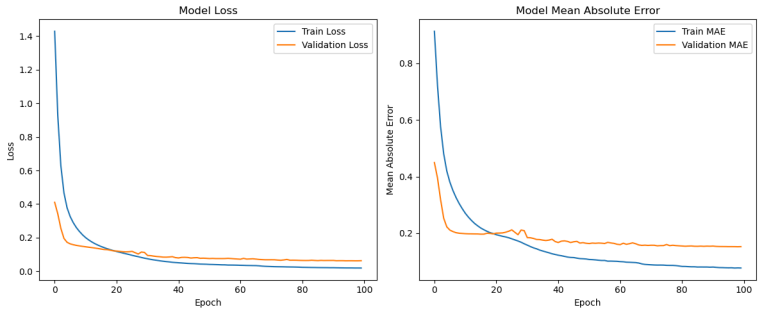

Figure 4: *Loss and MAE for train-validation model training*

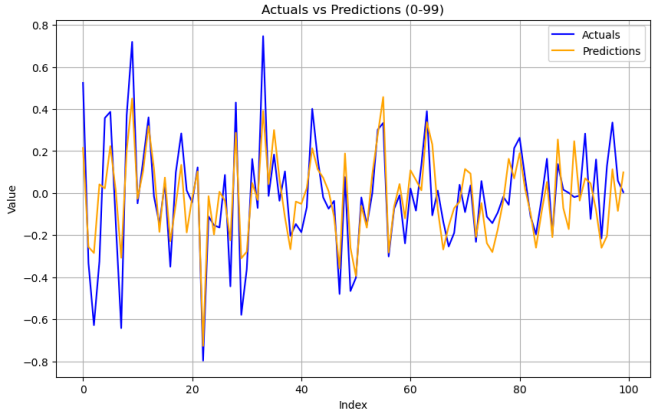

Figure 5: *Best-fit validation prediction from Spec2SpeVEC model (Metharbital: CCC1(CC)C(=O)N(C)C(=O)N=C1O)*

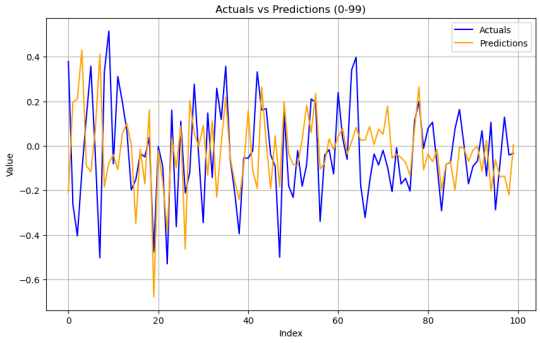

Figure 6: *Worst-fit validation prediction from Spec2SpeVEC model (5-Methylfurfuryl acetate: CC(=O)OCc1ccc(C)o1)*

Initial attempts to extend the model's capabilities to generate SMILES strings directly from the predicted molecular vectors proved challenging and impractical. The primary obstacle lay in the inherent complexity of reverse-engineering SMILES strings from the SPE-derived vector representations. While the SPE tokenization process effectively captures molecular

substructures in a continuous vector space, the inverse transformation presents significant challenges due to the loss of precise syntactic information during the embedding process.

EVALUATION

AutoEncoder

Training and validation occurred over 10 epochs using cross entropy loss. The AE successfully learned some patterns including opening and closing parenthesis, predicting characters that were in the input, and generating varying outputs corresponding to the length and whitespace of the input. We see this reflected in the predicted outputs in **Figure 7**. We can also see from the example outputs that there are characters that appear after a lot of whitespace and there are a lot of repeating characters still. The model may be improved by training on more SMILES sequences with increased kernel compute.

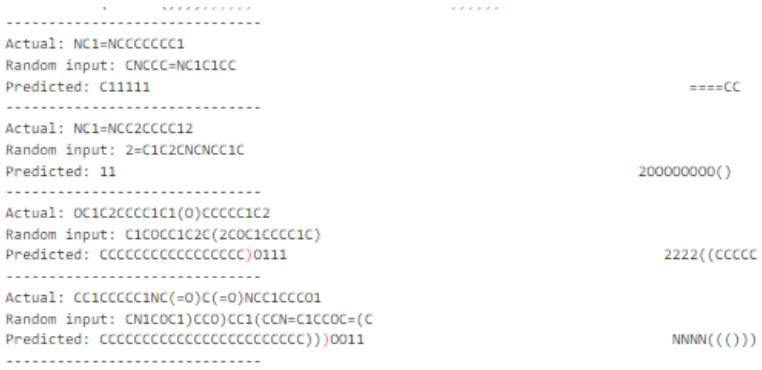

Figure 7: *Predicted SMILES AE example outputs from Epoch 10.*

Transformer

Unfortunately, with our limited computational resources, even with the cross-attention mechanism in our Transformer, our output was largely strings of carbon. This is due to the skewed dataset and our limited set of data that we used for training (~1500 datapoints). To evaluate the performance of the model, we examined the accuracy and F1 scores for the training and validation sets.

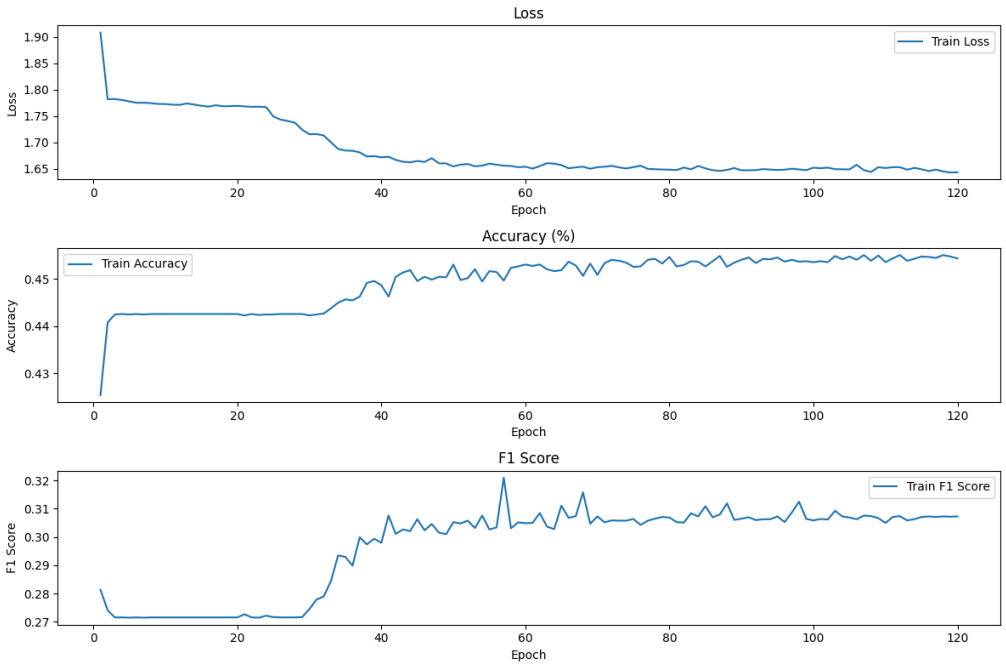

Figure 8 : *Training metrics for Transformer-only model over 120 epochs, with unique character SMILES tokenization.*

Additionally, we limited our batch size to a mere 16, which allowed the transformer to run without memory issues, but severely limited its capabilities. We can see this in the following figure, where our output after 120 epochs predicts only two different tokens in the target DeepSMILES sequence.

```
Epoch [120/200], Train Loss: 1.6432, Train Accuracy: 0.4544, Train F1: 0.3073
Original DeepSMILES: OC)COC))CCC=CC=CC=6)))))C
Predicted DeepSMILES: =CCCCCCCCCCCCCCCCCCCCCCCCC
Original DeepSMILES: O=CC=CC=CO)CO)=C6O)))))))C
Predicted DeepSMILES: =CCCCCCCCCCCCCCCCCCCCCCCCC
Original DeepSMILES: O=COCCCC)CCC6CC)C)))))))))C=C
Predicted DeepSMILES: =CCCCCCCCCCCCCCCCCCCCCCCCC
Original DeepSMILES: OCC=CC=CC=6C)))C)))C)))CCN)C
Predicted DeepSMILES: =CCCCCCCCCCCCCCCCCCCCCCCCC
Original DeepSMILES: O=CCC=CC=CC=6CC=CC=CC%13=6
Predicted DeepSMILES: =CCCCCCCCCCCCCCCCCCCCCCCCC
Original DeepSMILES: O=N=O)C=CC=CCC6=CC=CC=6CC9
Predicted DeepSMILES: =CCCCCCCCCCCCCCCCCCCCCCCCC
Original DeepSMILES: O=COCCC)C))))C=CC=CC=C6))C
Predicted DeepSMILES: =CCCCCCCCCCCCCCCCCCCCCCCCC
Original DeepSMILES: O=CCCCC=CC7C=C6))CC=CC=CC=6
Predicted DeepSMILES: ==CCCCCCCCCCCCCCCCCCCCCCCC
Original DeepSMILES: NC=NC=CC=6CC=CC=CC=6)))))))C
Predicted DeepSMILES: =CCCCCCCCCCCCCCCCCCCCCCCCC
Original DeepSMILES: O=COC))CN=CO)C)))CCC=CC=CC=6
Predicted DeepSMILES: =CCCCCCCCCCCCCCCCCCCCCCCCC
Original DeepSMILES: O=CC=CN=CN5C))))NC=O)N6C)))C
Predicted DeepSMILES: =CCCCCCCCCCCCCCCCCCCCCCCCC
Original DeepSMILES: O=COC))CCCCC=CCC=CCC=CCC=CCC
Predicted DeepSMILES: =CCCCCCCCCCCCCCCCCCCCCCCCC
```

Figure 9: *Final epoch of the unique-character DeepSMILES tokenization training loop, where the predicted output is still primarily Carbon.*

When using this trained model to predict DeepSMILES characters in our validation loop, the output was even worse.

```
Original DeepSMILES: O=COCCCC=CC=CC=6)))))))))CC)C
Predicted DeepSMILES: ============================
Original DeepSMILES: [H]C[H])[H])OC=CC=CC=C6)C=O)C
Predicted DeepSMILES: ============================
Original DeepSMILES: O=COCCC=CCC)))))))CC=CC=CC=6O
Predicted DeepSMILES: ============================
Original DeepSMILES: FCC=CC=CC=6))C=NC=CCBr)=N6
Predicted DeepSMILES: ============================
Original DeepSMILES: O=COCC=CCC=CC=CC=6)))))))))C
Predicted DeepSMILES: ============================
Original DeepSMILES: N=CCCCCCCC8)))))))CCCCCC6
Predicted DeepSMILES: ============================
The average validation loss is: 3.974201571941376
The validation accuracy is: 0.1260771006345749
The validation F1 score is: 0.02823152020573616
```

Figure 10: *Predicted DeepSMILES strings from the unique-character DeepSMILES tokenization validation loop. We see a high loss (3.94), low accuracy (12.6%) and an F1 score of only 0.028.*

Training metrics consisting of loss, accuracy, and F1 for the substring tokenization method are shown in **Figures #8-10**. True convergence was not observed for any of the metrics. Furthermore, the predicted smiles strings along with their corresponding theoretical strings were also monitored. Several of these DeepSmiles pairs are shown in **Table**. Despite increased training, the model continued to generate DeepSmiles that predominantly consisted of carbon characters. Thus, implementation of an alternative tokenization method did not improve performance of the transformer model.

Figure 11: *Accuracy generated over 50 epochs of training for the substring tokenization method.*

Figure 12: *Loss values generated over 50 epochs of training for the substring tokenization method.*

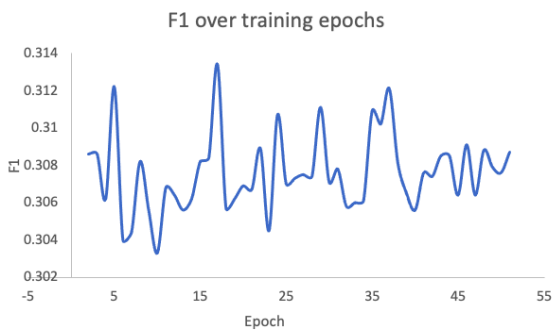

Figure 13: *F1 values generated over 50 epochs of training for the substring tokenization method.*

	actual	predicted
0	O=CN=CO)CF)=CN6COCCO))CO)C5	=CCCCCCCCCCCCCCCCCCCCCCCCCCC
1	O=COC))CCCCCCCC=CCC=CCC=CCC	==CCCCCCCCCCCCCCCCCCCCCCCCCC
2	O=CO)C=CC=CC=CC=C6))CF)F)F	=CCCCCCCCCCCCCCCCCCCCCCCCCC
3	O=COCC)))CCCCCCC)))))CCCCCCC	=C=CCCCCCCCCCCCCCCCCCCCCCCC
4	O=CCO)=CC=CC=C7CBr))))))CBr	=CCCCCCCCCCCCCCCCCCCCCCCCCC
5	O=COCCCC)C)))))CCCCCCCCCC	=CCCCCCCCCCCCCCCCCCCCCCCCCC
6	N#CC=NC=CC=N6)CC=CCF)=CC=6	=CCCCCCCCCCCCCCCCCCCCCCCCCC
7	O=CC=CC=COC))C=C6O)))))))C	CCCCCCCCCCCCCCCCCCCCCCCCCCC
8	O=COCC)))CCCCCCCC=CCCO)CCCCCC	=CCCCCCCCCCCCCCCCCCCCCCCCCC
9	CC=CC=CC=6))[Si]C)CC=CC)C5))C	=CCCCCCCCCCCCCCCCCCCCCCCCCC
10	O=COCC)C)C)))NCCCC5CO)CCCO	=CCCCCCCCCCCCCCCCCCCCCCCCCC

Table 1: *Theoretical DeepSmiles strings and their corresponding predictions generated from the substring tokenization method.*

POST-PREDICTION ANALYSIS

Post-prediction analysis was explored to predict SMILES strings beyond simple string comparison. While the generative SMILES string models discussed above were not successful in generating SMILES strings beyond just carbon chains, a script was developed to test the accuracy of prediction. Specifically, the script is implemented to evaluate two structure-based metrics: molecular weight difference and Tanimoto similarity using Morgan fingerprints. Using RDKit, invalid SMILES strings from predictions are filtered out by employing sanitization metrics. This filtering step is crucial to ensure only chemically feasible structures are analyzed.

The script calculates the exact molecular weight of the predicted and actual molecules for each valid prediction and the Tanimoto similarity between the actual and predicted molecules using Morgan fingerprints (ECFP4, radius=1, 2048 bits), which measures the structural similarity between molecules.

These metrics provide complementary views of prediction quality - while molecular weight difference captures the overall composition accuracy, Tanimoto similarity measures structural feature preservation. Together, they suggest that despite its relatively low validity rate, our model can capture both the compositional and structural aspects of the target molecules when it produces valid predictions.

DISCUSSION

We found this to be a challenging problem and attempted to approach it in varied ways. Our unbalanced dataset made the challenge even greater, as our models found it easiest to output Carbon strings for our heavily Carbon-based molecules. We were also severely limited by computational power and memory, as all the methods we explored were data-hungry and required complex architecture to produce valid results.

As mentioned, our biggest roadblock was that of our highly unbalanced data. Filtering to molecules with fewer carbons crippled our dataset, as the vast majority of molecules had a preponderance of carbon chains. Thus, the only way to fix this unbalance would have been to find or generate a new dataset with a greater variety of input molecules.

We considered several improvements to our models that would expedite optimization of these models in the future.

Given the appropriate computational feasibility, we would hypertune our model parameters. We used the Adam optimizer for our models, which is sensitive to initial learning rate. Tuning the learning rate might prevent the model from falling into “wells” of local minima. Additionally, we could test other optimization algorithms, such as stochastic gradient descent (SGD), where we can tune learning rate as well as momentum and weight decay. Momentum helps accelerate gradients for model convergence, while weight decay is another method of regularization.

In the model architecture itself, we can hypertune the number of attention heads, the dimensions of the model, and the number of encoder and decoder layers. We would also increase the batch size in the Transformer, as each training loop only utilized 16 training samples propagated through the model in one pass. With only 16 samples to compare against, the convergence ability suffered. Given that both our input and output are in complex forms, we would need an equally complex architecture to correctly address the problem at hand. Given infinite computational power, our models would have looked significantly different from their current state.

Because the model we were trying to recreate is so complex, we struggled with even getting a model that would adequately recreate SMILES from SMILES. In the future with a model that was closer to the target sequence, GridSearchCV would be the option of choice to assist in our hyperparameter tuning, but this also would add to the other problem we were facing

with the lack of compute power. We were unable to run our transformer and autoencoder to a satisfactory number of epochs and with enough MS inputs due to constraints on memory and time. Even with the use of resources like Google Colab or Kaggle, our code would stop short. One of the solutions attempted was to run the model for 50 epochs, which was within the memory constraints, save the model, then run the saved model for another set of training. This allowed our model to progressively improve towards the target sequences, however we were unable to converge on the true target. With more compute power we would have liked to increase the number of epochs per training run and decrease the batch size to converge on a model that could predict the SMILES sequences more accurately.

Another improvement upon our model would be evaluation of different SMILES tokenization methods. We tested language-based tokenization like the BartTokenizer from HuggingFace, which showed limited success. We then moved to creating custom tokenization dictionaries using our input dictionaries, both with unique characters and 4-character strings. This, too, had limited success. Our next step would be to test tokenization based on molecular functional groups, rather than character strings.

Functional groups are a much more logical and scientific breakdown of molecules that could help any version of the SMILES autoencoder (AE) or the Transformer to learn the proper structure of the molecules. This however does present challenges because molecules are not purely a collection of functional groups, our models could have trouble with sequences that are not dense with functional groups and therefore make poor predictions of those molecules.

We could also evaluate different tokenization of the GC-MS spectra. We could bin m/z abundance values, or use a model such as Spec2Vec to create intermediate embeddings.

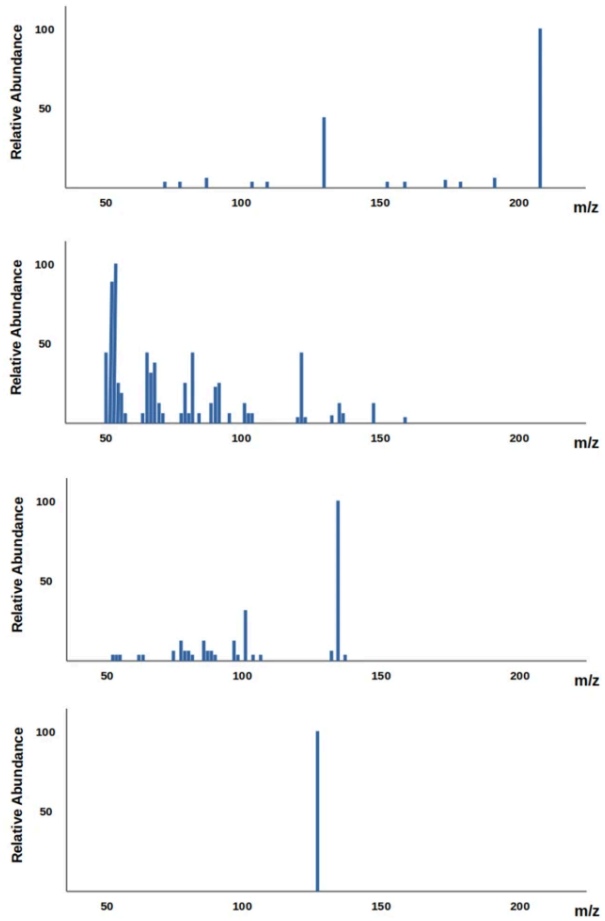

Figure 11: *Tandem mass spectra data for the same molecule under different experimental conditions. Image taken from https://www.nature.com/articles/s42004-023-00932-3#Sec19.*

As mentioned earlier, Mass Spec spectra are not always unique and multiple mass spectra can be used to define the same molecule (**Figure 11)**. We could also implement a beam search to produce the top *n* number of predictions and their corresponding probabilities. With the most likely candidates in hand, we could use another method of analysis to produce a closer prediction of the unidentified molecular structure. The `pytorch_beam_search` program by Juan Ramirez-Orta and Risang Baskoro appears to be promising as it can be used as a transformer to directly map from characters to target characters. However, this model performs integer mapping which was an issue for our SMILES AE; we found that the model tends to favor higher integer values. Potentially recreating the `pytorch_beam_search` with one-hot encoding instead would be advantageous for the MS uniqueness problem.

Overall, this problem gave us a taste of many different machine learning methods. Though we did not reach the accuracy and precision that we sought, the exposure to different models and the puzzle of pre- and post-processing has been a successful method of wrapping up our semester of learning about machine learning methods.

REFERENCES

Alammar, J. (n.d.). The illustrated transformer. https://jalammar.github.io/illustrated-transformer/

Baoilleach. (n.d.). GitHub - baoilleach/deepsmiles: DeepSMILES - A variant of SMILES for use in machine-learning. GitHub. https://github.com/baoilleach/deepsmiles

Convolutional Neural Network (CNN). (n.d.). TensorFlow. https://www.tensorflow.org/tutorials/images/cnn

Litsa, E. E., Chenthamarakshan, V., Das, P., & Kavraki, L. E. (2023). An end-to-end deep learning framework for translating mass spectra to de-novo molecules. *Communications Chemistry, 6*(1). https://doi.org/10.1038/s42004-023-00932-3

MassBank of North America. (n.d.). https://mona.fiehnlab.ucdavis.edu/

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

Ramirez-Orta, J., jarobyte91. (2022, June 3). PyTorch Beam Search. PyPI. https://pypi.org/project/pytorch-beam-search/

Sakar, A. (2023, August 23). Building a Transformer with PyTorch. *Datacamp.* https://www.datacamp.com/tutorial/building-a-transformer-with-py-torch

The annotated transformer. (2018, April 3). https://nlp.seas.harvard.edu/2018/04/03/attention.html

"Tf.keras.layers.Bidirectional | TensorFlow Core V2.3.0." TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Bidirectional

XinhaoLi. (n.d.). GitHub - XinhaoLi74/SmilesPE: SMILES Pair Encoding: A data-driven substructure representation of chemicals. GitHub. https://github.com/XinhaoLi74/SmilesPE