5-2023

Lignin Copolymer Property Prediction Using Machine Learning

Collin Larsen
University of Arkansas

Lignin Copolymer Property Prediction Using Machine Learning

Collin Lyn Larsen

Ralph. E. Martin Department of Chemical Engineering

College of Engineering

University of Arkansas

Fayetteville, AR

Honors Advisor: Dr. Keisha B. Walters

Graduate Assistant: David Chem

April 28, 2023

ABSTRACT

Lignin, an abundant biopolymer, is a waste byproduct of the paper and pulp industry. Despite its renewable nature and potential applicability in various products, such as plastics and composites, the development of lignin-based materials has been impeded by the cumbersome, Edisonian process of trial and error. This research proposes a novel approach to forecasting the properties of lignin-based copolymers by utilizing a recurrent neural network (RNN) based on the Keras models previously created by Tao et al. Example units of modified lignin were synthesized via esterification and amination functional group modifications. To increase the efficiency and accuracy of the prediction model, the Keras models were redeveloped as a PyTorch RNN model. The PyTorch RNN model produced similar or superior precision with a minimum decrease in execution time of 60%. Furthermore, the PolyInfo database was investigated for its potential use as a universal copolymer data set, and a complimentary interpreter was developed. While the interpreter produced a data set from basic copolymers, assembling the framework and composition of more intricate and ambiguous copolymers proved unreliable. Further exploration and development are required to construct a comprehensive copolymer data set from the PolyInfo database.

1. INTRODUCTION

Since the 1950s, plastic production has increased by ~6 million metric tons (MMt) annually, generating 110 MMt/yr – or ~75 lbs of plaster per person.[1] To date, only 7% of plastics produced have been recycled, with the remaining sent to be discarded or incinerated. The production of plastics has become so prominent that anthropologists have dubbed the current period as the "plastic age," and this trend is expected to continue to the point that plastic production will consume 20% of the global oil supply by 2050.[2,3] This trend is expected to have a significant environmental impact, claiming 15% of the annual greenhouse gas (GHG) allotments necessary to meet the 1.5 °C global temperature increase limit from pre-industrial levels.[3] However, despite currently being geared towards the petrochemical industry and subsequent creation of virgin polymers, a net-zero plastic economy is possible, with a potential cost-saving of $288 billion.[4] However, the lifecycle analysis (LCA) relies on current energy-intensive, economically unviable processing techniques that the Department of Energy identifies as "insufficient to address the growing accumulation" of plastic waste.[5] Additionally, their model requires carbon capture and utilization to offset the carbon-intensive processes, likely further impacting overall economic viability, highlighting the need for alternative polymers, processing strategies, and upcycling techniques.

At the same time, industrial agricultural processes such as paper, pulp, and biofuel manufacturing produce an abundance of low-value lignin - upwards of 50 MMt/year, which is often considered a waste byproduct and typically discarded or incinerated for low-quality heat production.[6] However, the abundant natural biodegradable biopolymer has the potential to be used in a wide array of products, such as plastics and composites. In recent practice, lignin has been utilized as a composite material rather than as an additive like in the past, with increasing molecular weight causing increased brittleness. Additionally, copolymers with a wide array of chemistries are feasible due to lignin's staggering complexity and numerous function groups, with molecular weights up to 20,000 g/mol.[7] Consequently, lignin-based thermoplastics and thermosets have been developed, with lignin coupled with PS, styrene, ethylene, and butylene (SEBS) with tensile strength and flexural moduli increasing with additional lignin incorporation as well as incorporation

in a variety of other polymers such as PP and polyvinyl chloride (PVC).[8,9] Most importantly, the incorporation of olefinic chains (such as those present in polyolefins like PP and PE) grafted to lignin demonstrate tunable thermal-mechanical properties, allowing the modification of fossil fuel derivatives while maintaining processability suitable for consumer applications.[10-12]

Material development typically requires the synthesis of the material before its properties can be accessed, imposing significant limitations on lignin-based copolymer development. By merging machine learning with material development, the Edisonian approach of trial and error can be circumvented, and insights can be gleaned into the properties of a material prior to synthesis. Therefore, this research aims to develop a neural network model that can be used to predict the properties of organic copolymers, specifically those of lignin, thereby adding impetus to lignin-based copolymer research and other bio-copolymers. Completing the model would accelerate the development, adoption, and commercialization of organic biopolymers and increase the utilization of lignin beyond a waste product, thereby simultaneously addressing two critical ecological concerns.

2. METHODOLOGY

2.1 Lignin Modifications

The model can only be used if some of the lignin-based units intended to be utilized in the predicted copolymers can first be synthesized. The two primary modifications we will be investigating are esterification and amination.

2.1.1 Esterification of Lignin. Lignin is dissolved in acetic anhydride in a 1:2 wt. ratio. For the catalyst, 0.01 mL of 1-methylimidazole per gram of lignin is used. The reaction is conducted at 50 °C for 3 hours in a stirring glass reactor. Upon completion, the solution is poured into cold deionized water to quench the reaction and form the esterified precipitate. The precipitate is filtered and washed with deionized water until the pH of the filtrate reaches about 5. Finally, the solid is dried in an oven at 50 °C.

2.1.2 Amination of Lignin. Phenolated lignin is dissolved in sodium hydroxide (0.4 mol/L) at a 1.0 g: 2.0 mL ratio for 10 minutes under continuous stirring. Ethylene diamine and formaldehyde are added into the solution at a ratio of 1.5 g ethylene diamine: 1.0 g phenolated lignin: 2.0 g formaldehyde. The mixture is kept at 100 °C for 4 hours with continuous stirring. The mixed product is then dialyzed using dialysis tubing with a molecular cut-off of 3000 Da. Finally, the solid is dried in an oven at 50 °C.

2.2 Dataset Development and Feature Selection

To develop machine-learning models, a dataset containing only the input data to the model is required. In this case, the input data are the structure data (x) and the property data (y). Two datasets are to be constructed, one with a large amount of general copolymer data and another with copolymer data specific to lignin and similar biopolymers. The primary dataset used in this study is the PolyInfo database, an extensive database containing the properties of several polymers and copolymers.[13] Regardless of the dataset, each can be broken into three sections: metadata, composition/structure, and properties.

Metadata columns typically contain a reference to the original source of the data and a unique sample identification number specific to the dataset. The composition/structure section has columns for the copolymer type, such as random, block, and alternation, as well as columns describing the ratio and structure of the monomers. Lastly, there is a column for each property recorded for that sample, with only one property – such as glass transition temperature – predicted per model.

The structure of the monomers is stored in different formats that are converted to a Morgan fingerprint, a mathematical representation of a molecule and its features in a binary vector.[14] One storage format is the Simplified Molecular-Input Line-Entry System (SMILES) notion, a form of line notion for describing the structure of a chemical species using short ASCII strings.[15] Using the RDKit library's Molecule class, a mol object is generated from the SMILES of the monomer unit, which is an object representation of the atoms and bonds of the molecule.[16] This object is a powerful digital representation of a molecule, allowing for helpful analysis features such as rendering a molecular drawing of the molecule. The mol object is then converted to a Morgan fingerprint. The Morgan algorithm recursively adds neighboring atoms and calculates a Morgan fingerprint, allowing the machine to understand molecular structures in finer detail than the nanoscale.

From there, the structure of the copolymers is assembled from the Morgan fingerprints and the composition data of the copolymer into a vector consisting of 100 monomers in the correct ratio and configuration.

2.3 Model Development and Transfer Learning

The model replicated in this work is a recurrent neural network (RNN) Keras model developed by Tao et al.[17] Keras is a simple and easy-to-use machine-learning library that lends itself to the fast creation of prototypes.[18] Due to Keras's high-level Application Programming Interface (API), complex models can be constructed with a few lines of code. However, PyTorch has been shown to exhibit better performance on large-scale models.[19] Moreover, PyTorch offers greater flexibility than Keras – boasting a more

complex, sophisticated library capable of more advanced tasks such as dynamic computation graphing.[20] To accommodate the vast complexities inherent in lignin copolymers and the anticipated non-linear handling, the Keras models developed in Tao et al.[17] must be redeveloped as PyTorch models.

Given the limited research on lignin copolymers, a dataset developed from scratch would likely be inadequate for understanding the complex structure-property relationships of lignin. One solution to this issue is transfer learning. This technique involves freezing the underlying structures of a model already trained on general data to prevent their weights from being updated during training and retraining on a smaller, more specific dataset, such as lignin.[21] In similar research, transfer learning has proven effective at polymer property prediction with as little as 28 training sets.[21] Therefore, transfer learning, in addition to the switch to PyTorch, is anticipated to be the optimal approach for modeling the complex relationships that lignin copolymers exhibit.

3. RESULTS AND DISCUSSION

3.1 Lignin Modifications

3.1.1 Esterification of Lignin

The successful esterification of stock lignin was confirmed via Fourier Transform Infrared (FTIR) spectroscopy, as depicted in Figure 1. The emergence of a new peak at 1757 cm^{-1} for phenolic and 1740 cm^{-1} for aliphatic carbonyl groups indicated the presence of the added ester group. Additionally, the appearance of a new peak at 1267 cm^{-1} corresponding to epoxy/carbonyl stretching and a decrease in the appearance of the hydroxyl group (3050 –3700 cm^{-1}) confirmed the esterification process.

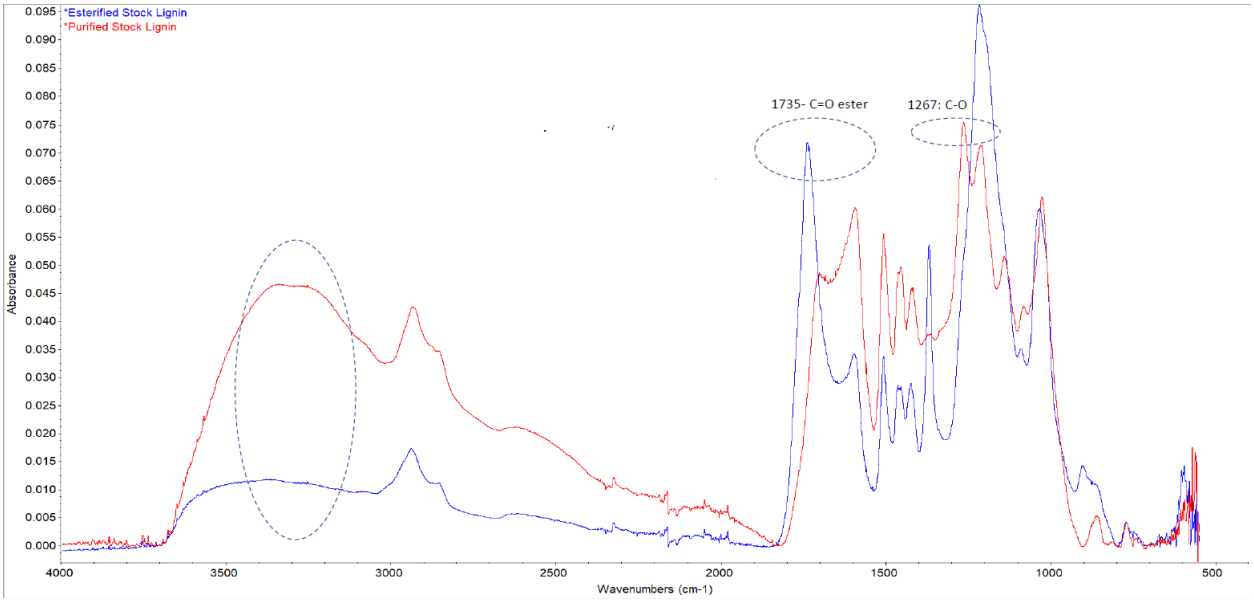

Figure 1: FTIR spectra of purified vs. esterified stock lignin.

3.1.2 Amination of Lignin

Amination of lignin was expected to be confirmed by FTIR spectroscopy; however, the appearance of the primary amine peak at 1650-1590 cm^{-1} is overshadowed by the aromatic stretching at 1594 cm^{-1}, as illustrated in Figure 2. Due to this overshadowing, the amination cannot be confirmed by FTIR. However, as demonstrated in Figure 3, confirmation of successful amination was obtained via X-ray photoelectron

spectroscopy (XPS) analysis. XPS analysis revealed an increase of nitrogen, confirming the synthesis of aminated lignin regardless of phenolation pretreatment.

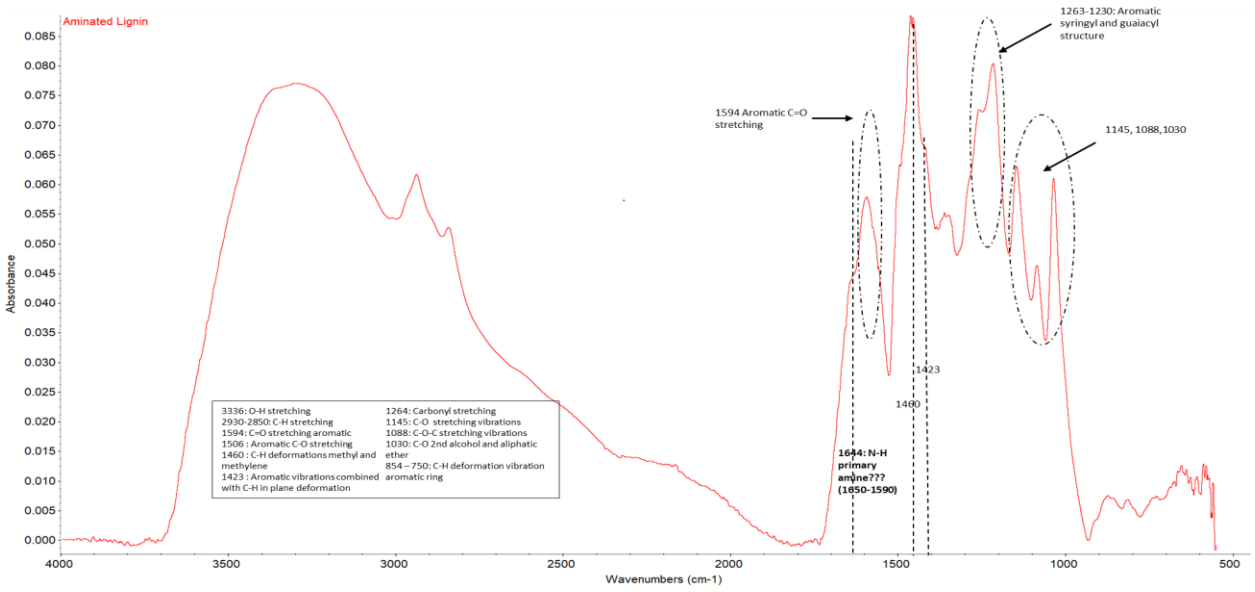

Figure 2: FTIR spectrum of aminated lignin.

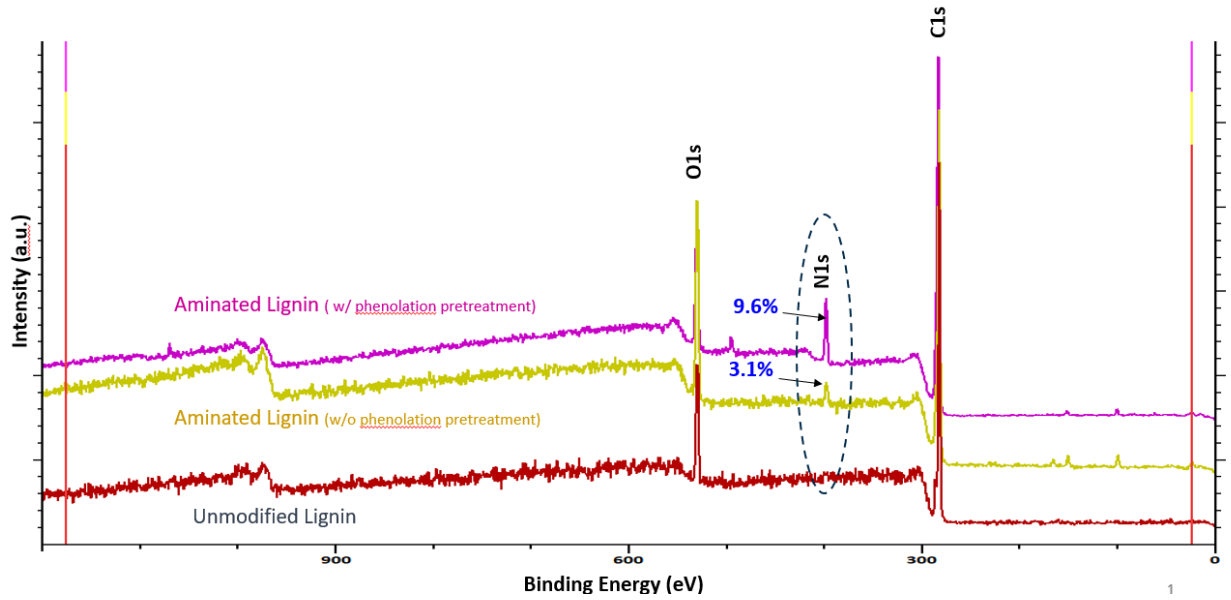

Figure 3. XPS characterization of unmodified lignin, aminated lignin synthesized from unmodified lignin, and aminated lignin synthesized from phenolated lignin.

3.2 Keras Breakdown and PyTorch Reconstruction

The Kera RNN structure code for datasets 2 and 3 from Tao et al.[17] are shown in Figures 4 and 5. Despite predicting very different properties (nuclear magnetic resonance (NMR) signal and glass transition temperature), the two models differ in lines 287 and 356 (Figures 4 and 5, respectively). These lines of code define the first layer of the RNN model as a bidirectional Long Short Term Memory (LSTM) with an input shape of 100 time steps with 80 and 125 features at each step for datasets 2 and 3, respectively. Thus, by reformatting the input shape to intuitive recognition by replacing the value of 100 with "X_train.shape [1]" and the value of 80 and 125 with "X_train.shape [2]", the two models become identical. With this change, a summary of the two models can be produced, as shown in Table 1.

```
281         # data split into train/test sets
282         X_train, X_test, y_train, y_test = train_test_split(Mix_X_100Block, DF['Tg_avg'], test_size=0.2, random_state=11)
283
284         # model setup using the optimized architecture
285         LSTMunits = 20 # hyperprameter for LSTM
286         RNNmodel = Sequential()
287         RNNmodel.add(Bidirectional(LSTM(LSTMunits, return_sequences=True), input_shape=(100,125)))
288         RNNmodel.add(Bidirectional(LSTM(LSTMunits, return_sequences=True)))
289         RNNmodel.add(TimeDistributed(Dense(int(LSTMunits/2), activation="relu")))
290         RNNmodel.add(Reshape((int(LSTMunits/2*100),)))
291         RNNmodel.add(Dense(1))
292
293         RNNmodel.compile(loss='mse', optimizer='adam')
294         RNNmodel.fit(X_train, y_train, validation_split=0.2, epochs=200, batch_size=64)
```

Figure 4. Keras RNN structure code – from train_datset3.py[17]

```
350         # data split into train/test sets
351         X_train, X_test, y_train, y_test = train_test_split(Mix_X_100Block, DF['19F NMR Signal-to-Noise Ratioa'].astype(np.float64), test_size=0.2, random_state=11)
352
353         # model setup using the optimized architecture
354         LSTMunits = 20 # hyperprameter for LSTM
355         RNNmodel = Sequential()
356         RNNmodel.add(Bidirectional(LSTM(LSTMunits, return_sequences=True), input_shape=(100,80)))
357         RNNmodel.add(Bidirectional(LSTM(LSTMunits, return_sequences=True)))
358         RNNmodel.add(TimeDistributed(Dense(int(LSTMunits/2), activation="relu")))
359         RNNmodel.add(Reshape((int(LSTMunits/2*100),)))
360         RNNmodel.add(Dense(1))
361
362         RNNmodel.compile(loss='mse', optimizer='adam')
363         RNNmodel.fit(X_train, y_train, validation_split=0.2, epochs=200, batch_size=64)
```

Figure 5. Keras RNN structure code – from train_dataset2.py[17]

Table 1. Architecture summary of the two Keras RNN models used in Tao et al.[17].

Layer (num)	Layer (type)	Features	Inputs
1	LSTM	Bidirectional	X_train.shape[2], lstm_units
2	LSTM	Bidirectional	lstm_units*2, lstm_units
3	Time Distributed	ReLU	Lstm_units*2, lstm_units/2
4	Reshape	None	Lstm_units/2 *X_train.shape [1], 1
5	Dense	None	1

For the first two layers, nn.LSTM is used to represent the LSTM layers in PyTorch. In PyTorch, bidirectional=True is passed to the initialization of the LSTM layers to establish them as bidirectional, and batch_first=True is passed to restructure the return order with the batch size first. Setting the batch size to return first simplifies the return structure to make it more like the Keras model.

In Keras, layer 3 applies a dense layer to each time step of the output sequence using a time-distributed layer. In PyTorch, the equivalent of a dense layer is the linear layer, which are both fully connected layers that perform a matrix multiplication before a bias addition to the input data. To apply the linear layer to each time step, the time-distributed layer is replaced with a sequential container with a linear layer that applies the same operation to each time step of the output sequence.

The reshape function in PyTorch replaces the reshape layer in Keras to flatten the vector. The main difference in developing this feature is that the Keras model must be given a "pre-flattened" vector, whereas the PyTorch reshape function can flatten intuitively. Finally, the last layer in the Keras model is a dense model with a single output. In PyTorch, this is replaced by a linear layer with lstm_units/2*X_train.shape[1]

input features and one output feature. Assembling this information into a working model, the RNNModel class code is produced (Figure 6).

```python
import torch
import torch.nn as nn

class RNNModel(nn.Module):
    def __init__(self, input_size, lstm_units): #input_size = 80 in dataset 2 and 125 in dataset 3
        super(RNNModel, self).__init__()
        self.input_size = input_size
        self.lstm_units = lstm_units
        self.lstm = nn.LSTM(input_size=input_size, hidden_size=lstm_units, bidirectional=True, batch_first=True) # batch at first dimension of input tensor
        #lstm output with 40 units with the same number of nuerons and settings
        self.lstm2 = nn.LSTM(input_size=lstm_units*2, hidden_size=lstm_units, bidirectional=True, batch_first=True)
        self.time_distributed = nn.Sequential(
            nn.Linear(lstm_units*2, int(lstm_units/2)),
            nn.ReLU()
        )
        self.fc = nn.Linear(lstm_units//2 * 100, 1)

    def forward(self, x):
        lstm_out, _ = self.lstm(x)
        lstm_out2, _ = self.lstm2(lstm_out)
        td_out = self.time_distributed(lstm_out2)
        reshaped = torch.reshape(td_out, (td_out.shape[0], -1))
        final = self.fc(reshaped)
        return final
```

Figure 6. Code of the RNN model class - “RNNModel” - developed in PyTorch.

Now that the model is constructed, the code that utilizes the model needs to be updated to match the model input and train the model. In Figure 6, the input, x, in the forward method of the PyTorch model is a tensor object, whereas the Keras model was capable of handling tensors and NumPy arrays. To correct this in the PyTorch code, the values output from train_test_split in Figures 4 and 5 must be converted to tensor objects, as demonstrated in Figure 7.

```python
# convert numpy arrays to PyTorch tensors
X_train_tensor = torch.from_numpy(X_train).float()
y_train_tensor = torch.from_numpy(y_train).float()
X_test_tensor = torch.from_numpy(X_test).float()
y_test_tensor = torch.from_numpy(y_test).float()
```

Figure 7. Code to convert the X and Y train and test data into tensor objects.

In Keras, the models were trained and optimized in two lines of code (293-294 and 362-363 in Figure 4 and 7, respectively), but in PyTorch, the model needs to be trained via “manual” iteration. To use the Adam optimizer and the Mean Squared Error (MSE) loss and train the model with 200 epochs, the code

shown in Figure 8 is utilized with the value of epochs set to 200. A variable "verb" was added to set the frequency at which the train and test loss are reported. When comparing the efficacy of the PyTorch model to that of the Keras model, the value of "verb" was set to 1 to mimic the output of the Keras model.

```
240     # create the optimizers
241     optimizer = optim.Adam(model.parameters(), lr=0.001)
242     criterion = nn.MSELoss()
243
244     # train the model
245     cur = time.time()
246     for epoch in range(epochs):
247         # set the model to train mode
248         model.train()
249         optimizer.zero_grad()
250         # forward pass
251         y_pred = model(X_train_tensor)
252         # calculate loss
253         loss = criterion(y_pred, y_train_tensor.unsqueeze(-1))
254         # backward pass and optimize
255         loss.backward()
256         optimizer.step()
257
258         # set the model to evaluation mode and evaluate
259         model.eval()
260         with torch.no_grad():
261             y_test_pred = model(X_test_tensor)
262             test_loss = criterion(y_test_pred, y_test_tensor.unsqueeze(-1))
263         if verb != 0 and epoch % verb == verb-1:
264             print(f"Epoch {epoch+1}, Train Loss: {loss.item():.4f}, Test Loss: {test_loss.item():.4f}")
```

Figure 8. Code to train the PyTorch RNN model.

Table 2. Results of the different RNN models for each dataset with 200 epochs and train-test loss reported every epoch. Execution times are evaluated using the same computer system one after another with no intensive process changes between execution.

200 Epochs Verb = 1	Keras – Dataset 2	PyTorch – Dataset 2	Keras – Dataset 3	Pytorch – Dataset 3
Execution time* (s)	154	61	89	35
Train R^2	0.78	0.79	0.90	0.92

Train MAE score	7.67	7.75	2.83	2.64
Train RMSE score	10.68	10.38	4.45	4.02
Test R^2	0.78	0.79	0.85	0.88
Test MAE score	9.04	8.59	4.24	4.14
Test RMSE score	10.67	10.42	6.52	5.65

Table 2 demonstrates that, in both cases, the PyTorch model was trained in significantly less time than the Keras model while achieving comparable or superior accuracy over the Keras model. Furthermore, it is essential to note that, for this data, the PyTorch model was running at a bottlenecked speed, and this data does not express the fastest computational speed of the PyTorch model. This is because the Keras model automatically outputs the train loss and test loss for each epoch, whereas the PyTorch model must be “told” when and how to report the train and test loss. For the data reported in Table 2, the value of "verb" was set to 1, meaning that the PyTorch model was reporting the train and test loss every epoch to match the reporting speed of the Keras Model. While it is advantageous to monitor the train and test loss associated with the training, reporting every epoch can take a significant amount of the calculation time and be unnecessary for understanding the effectiveness of the training. For example, running the code on dataset 2 with a reporting frequency every 20 epochs reduces the execution time to 48 seconds, and running the code on dataset 3 with a reporting frequency of every 100 epochs reduces the execution time to 23 seconds.

While the Keras model allowed for a straightforward, compact code for training the model, PyTorch requires manual iteration and more code. However, the PyTorch model is faster, taking less than half the time to execute compared to the Keras model. This comparison shows that the choice of deep learning framework can impact both the development time, the computation speed, and the overall

performance of the model, demonstrating that the PyTorch model is a better model to base future development.

3.3 Database Building

To compile a general copolymer database, the PolyInfo database was utilized, which contains data entries for numerous polymers and copolymers.[13] Following PolyInfo policies, each datapoint was manually collected and compiled into a table of raw descriptions called "Desc," along with a zip file of RDKit mol objects. To process this information, a PolyInfo interpreter code was developed.

As illustrated in Figure 9, the raw descriptions contain various information such as the sample IDs (SID), polymer IDs (PID), polymer types (ptype), component IDs (componentIDs), formula weights (fws), properties, and any related info (relinfo). A sample class was constructed to represent each sample, and the information was parsed from the description and assigned to each sample object. After extracting the zip file, the polymer ID was used to assign mol files to each sample data. In the interpreter code, the value of componentIDs is initialized with a raw string of component IDs taken from the description. As these IDs are used to assign mol files to the sample data, the parsed value of componentIDs is returned and reassigned. From here, the composition is calculated from the formula weights and the raw string of properties parsed as a dictionary of properties.

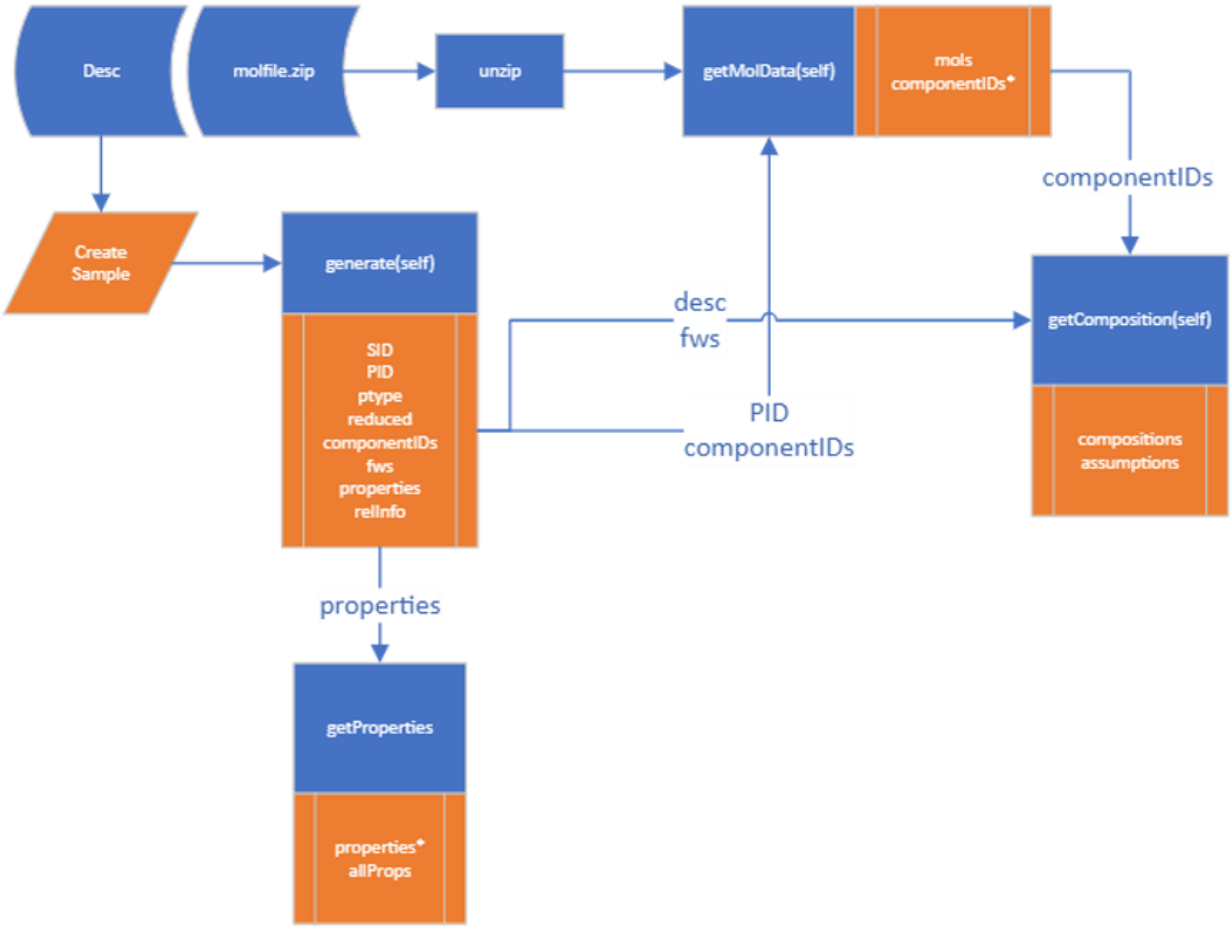

Figure 9. PolyInfo interpreter code flow diagram.

Lastly, a function was used to convert the list of sample objects into a pandas data frame of the samples, which could be exported as a spreadsheet. Figures 10 and 11 are molecules found in two samples collected from the interpreter code; the molecules are presented in their post-reaction partial monomer form with '*' notating a polymerization terminal. When evaluating the results of the interpreter, some issues began to stick out with the results immediately. For example, the sample represented in Figure 10 consists of a graft copolymer of copolymer ethene-co-propene and poly(2,5-dioxotetrahydrofuran-3,4-diyl). While the interpreter was able to correctly assemble the data of this sample - including separating the composition, there are some issues with how to assemble this structure. According to the description data, this sample has a composition of 99.58% ethene-co-propene and 0.42% poly(2,5-dioxotetrahydrofuran-3,4-diyl). When

training the model, each vector feature must represent a single, whole monomer unit. This means that to model this molecule, a vector built of at least 239 monomer units would need to be passed to the model to capture one unit of monomer B. Even this would only begin to capture the true detail of the composition. Another source of confusion for assembling this copolymer into a vector is that it is unclear whether "co" represents a block copolymer or an alternating copolymer and whether the 2,5-dioxotetrahydrofuran-3,4-diyl should be represented by one unit or if it should be represented as a block.

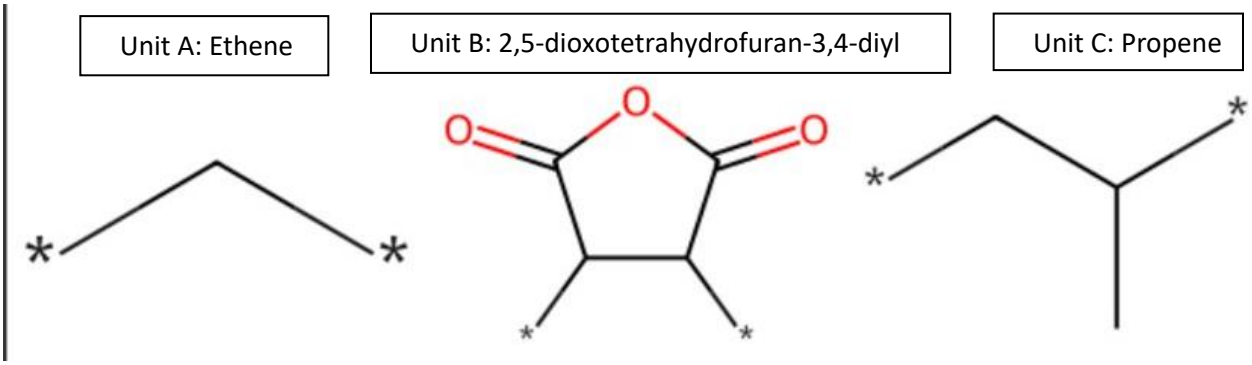

Figure 10. Monomer units of Poly[ethene-co-(prop-1-ene)]-graft-poly(2,5-dioxotetrahydrofuran-3,4-diyl) in their post-reaction partial structure forms

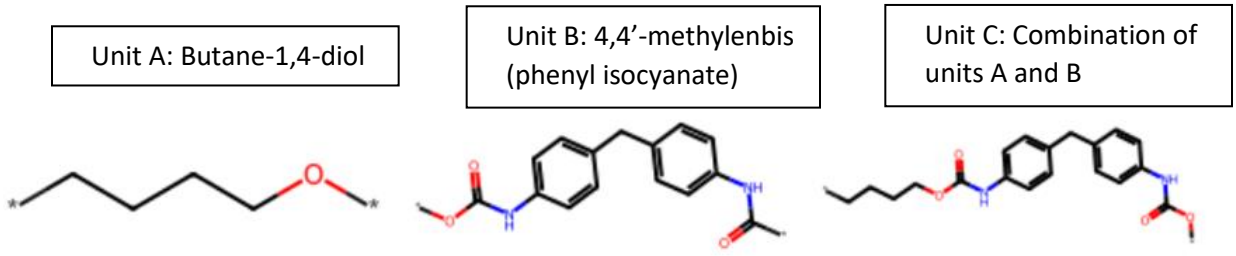

Figure 11. Monomer units of poly{[poly{(butane-1,4-diol);butane-1,4-diol]-alt-[4,4'-methylenebis(phenyl isocyanate)]} in their post-reaction partial structure forms

In the sample shown in Figure 11, the interpreter assumes that the polymer is constructed of three unique monomer units: A, B, and C. However, unit C represents the alternating copolymer butane-1,4-diol-alt-4,4'-methylenebis(phenyl isocyanate) in its post-reaction partial form. The composition of this copolymer is listed as 27% unit B and 73% unit C. Because there are three monomer units but only two compositions, the interpreter assumes that the composition of the monomer unit A needs to be derived and attempts to separate the composition to account for three unique monomer units, causing unit A to be assigned a non-zero composition – as it did with the sample in Figure 10. Instead, the interpreter must be adjusted to identify cases like this and assign a zero composition to unit A.

4. CONCLUSIONS

In conclusion, lignin-based units anticipated to be utilized by the final model were successfully synthesized in the lab by modifying lignin functional groups via esterification and amination. FTIR analysis confirmed successful esterification, while successful amination was confirmed via XPS analysis due to an overshadowing of the primary amine peak by the aromatic stretching in the FTIR results.

However, the PolyInfo database interpreter developed for data formatting proved too variable for consistent data collection. While the interpreter correctly assembled the values from the dataset for the typical copolymer, calculating the composition from the data proved unreliable for some samples with unusual and ambiguous compositions. As a result, many samples require manually reviewed to construct the correct end shape of the vector, making assembling the feature vector sets for each model a very labor-intensive task, likely necessitating a unique code for each assembly type. Therefore, exploring how to recursively transform the PolyInfo data into usable structures for the model and other potential data sources that could be collected quicker is necessary. These issues suggest that there may be limitations to this approach used, particularly in cases where the structures are ambiguous or complex.

Furthermore, it was discovered that the primary difference between the two Keras models developed by Tao et al. was only the size of the input vector, enabling the development of a universal PyTorch model using nn.LSTM, linear layers, and PyTorch's reshaped function to match the function and architecture of the Keras model. Transferring the Keras model to a PyTorch model proved successful, reducing calculation speed by at least 60% while maintaining accuracy in the same number of epochs. This indicates that the PyTorch model has better scalability and confirms the hypothesis that a PyTorch model will have improved calculation speeds.

This research provides valuable insight into creating a machine-learning model that can predict complex copolymer properties such as those from lignin. Despite the limitation in the approach to constructing the copolymer database, these findings can serve as a foundation for future research into not

only the area of lignin-based copolymer property prediction but general polymer development and machine learning incorporation in material science.

5. FUTURE WORK

This work is the first step in predicting lignin-based copolymers using machine learning; however, there are still many more steps to make that come to fruition. First, a generalized data set needs to be developed by correcting the PolyInfo interpreter and manual sample review or adopting a new database. Second, the efficacy of transfer learning should be evaluated using the PyTorch RNN model developed in this work trained on the generalized dataset transferred to learn the Tao et al. dataset 3. After validating the transfer learning code, the generalized model should be retrained on a small lignin copolymer dataset via transfer learning. Then, the results of that model should be evaluated against the lab data. Finally, the model's results can be improved by increasing the data points in the lignin dataset.

More than just predicting lignin-based copolymers, adopting this work to a generalized copolymer data set can serve as a foundation for the future of polymer property prediction by repurposing the general model for any specific polymer task using transfer learning.

6. ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my research mentor, Dr. Keisha B. Walters, and graduate mentor, David Chem, for their invaluable support and guidance throughout my research journey. Dr. Walters provided me with extensive guidance, encouragement, advice, and her expertise and insights were instrumental in the success of this project. David was crucial in the development of the methodology and execution of the lab modifications. He acted as my immediate go-to throughout the project, and his exceptional assistance and constant availability were priceless.

I would also like to thank the Honors College for their generous financial support, which has made this project possible. Their investment in me and my research enabled me to pursue this endeavor, providing me with the resources and opportunities to reach my goals.

Finally, I sincerely appreciate everyone who supported me in this journey, directly or indirectly. Most notably, I would like to thank the researchers of Tao et al. for their foundational work in the field of copolymer property prediction.

7. REFERENCES

1. Geyer, R.; Jambeck, J. R.; Law, K. L., 'Production, Use, and Fate of All Plastics Ever Made,' *Science Advances* **2017**, 3:e1700782.

2. Porta, R., 'Anthropocene, the Plastic Age and Future Perspectives,' *FEBS Open Bio* **2021**, 11:948-953.

3. Agenda, I. In The New Plastics Economy Rethinking the Future of Plastics, World Economic Forum, January 2016.

4. Meys, R.; Kätelhön, A.; Bachmann, M.; Winter, B.; Zibunas, C.; Suh, S.; Bardow, A., Achieving Net-Zero Greenhouse Gas Emission Plastics by a Circular Carbon Economy. Science 2021, 374, 71-76.

5. Britt, P. F.; Coates, G. W.; Winey, K. I. Report of the Basic Energy Sciences Roundtable on Chemical Upcycling of Polymers; 2019.

6. Rozite, L.; Varna, J.; Joffe, R.; Pupurs, A., Non-linear Behavior of PLA and Lignin-Based Flax Composites Subjected to Tensile Loading. J. Thermoplastic Comp. Mater. 2013, 26, 476-496.

7. Doherty, W. O.; Mousavioun, P.; Fellows, C. M., Value-Adding to Cellulosic Ethanol: Lignin Polymers. Ind. Crops Prod. 2011, 33, 259-276.

8. Reza Barzegari, M.; Alemdar, A.; Zhang, Y.; Rodrigue, D., Mechanical and Rheological Behavior of Highly Filled Polystyrene with Lignin. Polymer Comp. 2012, 33, 353-361.

9. Pouteau, C.; Baumberger, S.; Cathala, B.; Dole, P., Lignin–Polymer Blends: Evaluation of Compatibility by Image Analysis. Comptes rendus biologies 2004, 327, 935-943.

10. Wang, S.; Liu, W.; Yang, D.; Qiu, X., Highly Resilient Lignin-Containing Polyurethane Foam. Ind. Eng. Chem. Res. 2018, 58, 496-504.

11. Xiong, S.-J.; Pang, B.; Zhou, S.-J.; Li, M.-K.; Yang, S.; Wang, Y.-Y.; Shi, Q.; Wang, S.-F.; Yuan, T.-Q.; Sun, R.-C., Economically Competitive Biodegradable PBAT/Lignin Composites: Effect of Lignin Methylation and Compatibilizer. ACS Sus. Chem. Eng. 2020, 8, 5338-5346.

12. Abdelwahab, M. A.; Misra, M.; Mohanty, A. K., Injection Molded Biocomposites from Polypropylene and Lignin: Effect of Compatibilizers on Interfacial Adhesion and Performance. Ind. Crops Prod. 2019, 132, 497-510.

13. Polymer Database (PolyInfo) – DICE :: National Institute for Materials Science. https://polymer.nims.go.jp/en/ (accessed 2022-10-06).

14. Morgan, H. L. The Generation of a Unique Machine Description for Chemical Structures-a Technique Developed at Chemical Abstracts Service. Journal of Chemical Documentation 1965, 5 (2), 107–113.

15. Weininger, D. SMILES, a Chemical Language and Information System: 1: Introduction to Methodology and Encoding Rules. J Chem Inf Comput Sci 1988, 28 (1), 31–36. https://doi.org/10.1021/CI00057A005/ASSET/CI00057A005.FP.PNG_V03.

16. Rdkit.Chem.rdchem module¶. https://www.rdkit.org/docs/source/rdkit.Chem.rdchem.html (accessed Nov 2, 2022).

17. Tao, L.; Byrnes, J.; Varshney, V.; Li, Y. Machine Learning Strategies for the Structure-Property Relationship of Copolymers. iScience 2022, 25 (7), 104585. https://doi.org/10.1016/J.ISCI.2022.104585.

18. Keras. https://keras.io/ (accessed Nov 2, 2022)

19. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Köpf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv December 3, 2019. https://doi.org/10.48550/arXiv.1912.01703.

20. Pytorch. https://pytorch.org/ (accessed Nov 2, 2022).

21. Yamada, H.; Liu, C.; Wu, S.; Koyama, Y.; Ju, S.; Shiomi, J.; Morikawa, J.; Yoshida, R. Predicting Materials Properties with Little Data Using Shotgun Transfer Learning. ACS Cent Sci 2019, 5 (10), 1717–1730.