Haiku Generation

Akriti Gupta
akritig@seas.upenn.edu
Namita Shukla
nashukla@seas.upenn.edu
Bhumika Singhal
bhsingha@seas.upenn.edu
Smruti Chourasia
smrutich@seas.upenn.edu

Abstract

Haikus are known for their ability to paint a vivid picture in just a few words regarding a particular topic. They are 3-line long structured poems and they comprise of a fixed syllable scheme (5-7-5). Hence, Haiku generation is an example of a strictly constrained task. In addition, Haikus stand out amongst other poetry forms due to their flexibility in terms of topic, rhyme, and concise syntax. We have implemented various models published in the literature and their extensions to be able to generate good Haikus. These model implementations include SpaCy, novel CharRNN using LSTM, CharRNN using LSTM + Syllable structure and exploring variants of GPT (like GPTJ). Since, the task involves the generation of text legible to humans, the performance of the models is being evaluated by the quality of Haikus generated. For this, we are using GRUEN as the quality metric and average syllable count per line to evaluate the structure of haiku. The best results were obtained using Fine-tuned GPT-J with the max GRUEN score of 0.544 and the mean syllable structure of [4.9, 7.05, 5.10].

1 Introduction

Haiku is a short poem with an exceptionally well-defined structure. Traditional compositions consist of three lines following a 5, 7, 5 syllable pattern, typically contain a kigo, or seasonal reference, and often a kireji, also known as metrical pause. Such characteristics of Haiku make their automatic generation an extremely interesting NLP/Computational Linguistics task due to this strictly constrained nature. Also, since this is a text generation task, its evaluation is subjective in nature adding on to the complexity of the task.

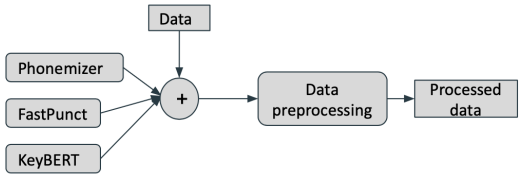

Figure 1: Model Architecture

The problem statement involves gathering of required data which is poetic in nature and preferably haikus, performing data cleaning using FastPunct, counting syllables using Python lib of Phonemizers and topic extraction using KeyBert (refer Figure 1). This data is then used to train models which are required to learn the character sequences, the association of the

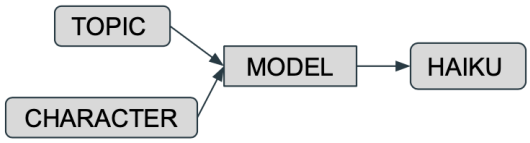

Figure 2: Model Architecture

words amongst themselves and with the syllable structure provided. The intended output of the models given an initial character and/or additional information such as syllable/topic is 3 lines of text which is coherent and has a poetic value while retaining the average number of syllables per line as close to the 5-7-5 structure as possible. Further, to evaluate the quality of the text generated, we are using GRUEN as our evaluation metric which judges the creative text produced based on its grammar, non-redundancy, focus, structure and coherence of generated text and hence is a better metric than BLEU.

It is this complex and creative nature of the problem statement that compelled us to take it up as our final project.

2 Literature Review

In [1], the authors inspired by neural language models and predicate argument relation explored various techniques for generating Haiku to share the culture of Haiku among young people using a emotional chatbot named Rinna. Firstly, they experimented using minimal vanilla recurrent neural network for learning character level neural language models. They further explored multi-layer RNN using LSTM and GRU for character-level language models. Following this they used Character Level Recurrent Convolutional Neural Network (RCNN) and Sequence Generative Adversarial Networks. They scraped various Haiku websites and also used Rinna(chatbot) query log for training the above mentioned models. The author's training set comprised on 100,000 Haikus, validation set and test set had 5000 Haiku's respectively.The models were hypertuned parameters like epoch and embedding size. The conclusion of these experiments were that RNN-LSTM performed better than the other models for web scraped Haikus whereas RCNN performed the best for Rinna generated Haikus.They were able to successfully launch the Haiku model in Rinna and obtain more than 50 million accesses in a few months.

In [2], the authors - Jack Hopkins and Doewe Kiela produced several models for generating poetry. They majorly focused on sonnets. They proposed two novel methodologies. The first approach used a neural language model(LSTM) which was trained on phonetic encoding catered to learn an implicit representation of both form and content of English Poetry. This model could learn basic poetic features like rhythm, alliteration and rhyme.
The second approach, was a character-level model which could produce more coherent text, but did not have any constraints on the form. They constrained the text at sample time by using a discriminator that would reject text that didn't conform to the desired meter (syllable format of 5-7-5). Their results were considerable good. An test, where participants were asked to classify a randomly selected set of human "nonsense verse" and machine-generated poetry, showed generated poetry to be indistinguishable from that written by humans.

The paper [3] introduces a model "Haikoo"

which is a transformer based model that outperforms previous SOTA neural network based haiku poetry generators. This model contains a GPT-2 model, which was fine-tuned on haiku poems and is able to successfully generalize the qualities of haiku poetry while retaining flexibility to compose poems on entities never seen on the training data, and a Plug and Play Language Models, which was used to control the generated results further than the classic prompt approach towards a particular topic / context. Hence, Haikoo as a model is able to generate haikus which satisfy poetic constraints and range from topics which are lyrical to hilarious. Vanilla LeakGAN model was used a baseline. TextGAN and GPT-2 (+ PPLM [generation with bag-of-words approach for highly focused control over the output.]) were the models fine tuned with task specific data and tested against the baseline. The metrics used to evaluate the generated haikus were Perplexity and "Sensicality" (how much sense it made), "Wisdom" (stupidity of a poem) and "Overall Quality" assessment by AMT workers. Using Bigram Perplexity scores on a sample output of both models it was found that GPT-2 performs better than GAN approach. On each AMT metric as well, GPT-2 outperforms the baseline with a particularly high margin in the metric of "Sensicality".

3 Experimental Design and Experimental Results

3.1 Data

Table enlists the sources of our data which we merged. Since we collected from different places, we conducted preprocessing on the data to ensure the structure and quality of text is correct. We used FastPunct module to add uppercase letters and punctuation , counted syllables using Python lib of Phenomizers and extracted topic using KeyBert. This helps with the text quality analysis.

Post this processing, we can see that out of total of 99387 haikus, 58239 haikus have the structure of 5-7-5 which makes about 60% of the dataset.

Source	**Size**
Tempes Libres(majority from Twitter) [4]	110K
Sam Ballas' PoetRNN corpus (dailyhaiku.org) [5]	8000
Herval Freire's Haikuzao corpus [6]	6000
Haiku Dataset from Kaggle (reddit.com/r/haiku) [7]	11269

Table 1: Data Sources

This is a text generation task and hence we have split the Haikus into a training set and development set. The train set has 123524 haikus and development set has 30882 haikus. The development set is used to test for overfitting in the models.

At the test time, we will pass in a "character" and / or syllable structure to the model and then the generated text (haikus) will be evaluated for their text quality and structure. The data sets used, their sources and counts are mentioned in Table 1.

3.2 Evaluation Metric

We conducted some preprocessing to ensure the structure of text is correct using FastPunct module to add uppercase letters and punctuation. It helps with the text quality analysis. Our proposed metric has 2 components, namely quality and structure.

3.2.1 Quality Metric

Although BLEU works well for assessing text quality based on the expected result, it doesn't help determine the quality of creative writing. So we used an automated system called GRUEN to assess the quality of the Haiku used in the dataset. To date, these metrics have focused almost exclusively on the content selection aspect of the system output, ignoring the linguistic quality aspect altogether. We bridge this gap by proposing GRUEN for evaluating Grammaticality, non-Redundancy, focus, structure and coherence of generated text. The GRUEN score is based on four criteria for text evaluation:

1. **Grammar:** A high grammatical score means the system output is readable, fluent, and grammatically
2. **Non-redundancy:** Non-redundancy refers to having no unnecessary repetition (e.g., the repeated use of name of a person when a pronoun would suffice).
3. **Focus:** A focused output should have related semantics between neighboring sentences
4. **Structure and coherence:** A well-structured and coherent output should contain well-organized sentences, where the sentence order is natural and easy to follow.

The final linguistic quality score is a linear combination of the above four scores: GRUEN-Score = grammatically + non-redundancy + focus + structure and is on a scale of 0 to 1.

3.2.2 Structure Metric

The haiku's generated are 3 lines of poem where each line should have 5,7,5 syllables respectively - this is the essence of a haiku. Hence for the generated poem we check the number of syllables in each line of the poem and plot the statistics (like mean, median etc.) across line 1 of each poem, then line 2 of each poem and same with line 3. This gives us the sense of structure.

3.2.3 Complete Evaluation

Evaluation score = GRUEN score + Syllable structure

Finally a good model will have a combination of high GRUEN score and a syllable counter with average close to 5, 7, 5 for respective lines.

3.3 Simple baseline

3.3.1 Methodology - SpaCy Matcher

SpaCy is a NLP open-source library. It is used in designing systems for information extraction or natural language understanding systems.

1. Our approach involved using the English words and their corresponding POS(Part-Of-Speech) tags using the Matcher module of SpaCy.
2. Then we defined the syllable(Haiku) structure i.e. 5-7-5 and the POS tag that we expect for each word in the Haiku.
3. Following this, we randomly generated words from our dataset that corresponded to the pattern specified.
4. This model is a simple guessing technique. It can be seen that there is no meaning of the Haikus generated and is made up of the most frequent words in the dataset. For example - randomly mixing sentences to generate a haiku with the required syllable structure.
5. Their quality is low, though their syllable structure is correct which is expected as we specified the pattern.

3.3.2 Results

We generated about 1500 poems -

1. **GRUEN score statistics** The average GRUEN score was 0.29 - with a median of 0.26 showing an almost consistent distribution. The max GRUEN score achieved was 0.79 for the poems generated. Hence the quality of the poems generated are not great. (Table 2)

Mean	0.2928
25%	0.1903
50%	0.2602
75%	0.3753
Max	0.7938

Table 2: Gruen Score of Spacy

2. **Syllable structure statistics** The average number of syllables across line 1: 4.9, line 2: 6.8 and line 3: 4.9 although the median of the distribution is line 1: 5, line 2: 7 and line 3: 5 - showing the generated data has learnt the structure of the data. This is simply because the Matcher module is able to generate based on the given POS tags and poems. (Table 3)

	Line 1	**Line 2**	**Line 3**
Mean	4.91	6.78	4.92
25 %	5.00	7.00	5.00
50 %	5.00	7.00	5.00
75 %	5.00	7.00	5.00
Max	7.00	8.00	8.00

Table 3: Syllable Structure Score of Spacy

3.3.3 Sample Haikus

Some sample haikus generated can be seen in Table 4.

Sample Haiku	**Meter**	**Quality**
Groggy mists submerge Slowly seasons salty seas Last sounds of nature	**5, 7, 5**	**0.398**
On a dreary shelf Slowly fade into orange Hard shapes of fish	**5, 7, 5**	**0.3441**
Pale afternoon sun With changes and mixtures Whispers and screaming	**5, 7, 5**	**0.2698**

Table 4: Example Haikus from Score-Spacy

3.4 Strong Baseline

3.4.1 Methodology - Character RNN

We'll talk about how we implemented and experimented with character-level RNN as discussed in papers [1] and [2] in our literature review. In a gist, our character level RNN reads haikus as a series of characters outputting a prediction and "hidden state" at each step, feeding its previous hidden state into each next step.
Step by step approach:

1. First step was data preprocessing - only the haikus (character by character) are feeded into the model without any additional metadata (such as syllable structure, topic of haiku,etc.).

2. Our neural network does not understand text so we have to convert our text data to integer. For this purpose we create a token dictionary and map character to integer and vice versa. In addition to this, we one hot encode the data to represent characters.

3. Now finally, we create a class called CharRNN. Here we use LSTM as our RNN model.

```
CharRNN(
  (lstm): LSTM(117, 512, num_layers=2, batch_first=True, dropout=0.5)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=512, out_features=117, bias=True)
)
```

Figure 3: Char RNN Architecture

Hyperparameters for the model and training are as follows:

- Number of LSTM layers = 2
- Number of epochs = 30.
- Dropout = 0.5
- Learning rate = 0.001
- Batch size = 128
- Sequence Length as 100
- Gradient clipping = 5
- Adam Optimizer
- Cross Entropy Loss as criterion

4. After training, we define a sampler (top-k samples with k =2) which takes in a character as a seed and generates next characters.

5. The generated output is stored in 3 columns in a csv where each column corresponds to each sentence in the Haiku. For evaluation, we use the GRUEN script.

6. The quality of generated Haikus is better than the simple baseline although since this learns Haikus on character to character basis the syllable structure is on the weaker side.

3.4.2 Results

We generated about 1500 poems -

1. **GRUEN score statistics** The average GRUEN score was 0.3396 - with a median of 0.2828. The max GRUEN score achieved was 0.8821 for the Haikus generated. Hence the quality of the Haikus generated is better than Spacy. (Table 5)

Mean	0.3396
25%	0.2162
50%	0.2828
75%	0.4252
Max	0.8821

Table 5: Gruen Score of CharRNN

2. **Syllable structure statistics** The average number of syllables across line 1: 7.58, line 2: 8.71 and line 3: 8.03 although the median of the distribution is line 1: 5, line 2: 7 and line 3: 6 - showing the generated data has learnt the structure of the data better.(Table 6)

	Line 1	**Line 2**	**Line 3**
Mean	7.58	8.71	8.03
25 %	5.00	7.00	5.00
50 %	5.00	7.00	6.00
75 %	10.00	10.00	11.00
Max	35.00	26.00	31.00

Table 6: Syllable Structure Score of CharRNN

3.4.3 Sample Haikus

Some sample haikus generated can be seen in Table 7

3.5 Extension to Strong Baseline

3.5.1 Methodology - Character RNN + Syllable Structure

As discussed in our MS2 results for our strong baseline (Char RNN), even though the output had good poems, they lacked the structural essence of Haikus. Hence, the key idea here is to extend the Character Level RNN network (our strong baseline) by feeding in the syllable structure information into it. The methodology is as follows -

Sample Haiku	Meter	Quality
Through swaying green arch Lucidity builds quickly Autumn may bring hope	**7, 8, 5**	**0.287**
Angry swollen spots Reflective yet transparent of a thousand suns	**4, 7, 8**	**0.327**
Hungry this morning Nature reclaimed our old Rapid eye movement	**7, 6, 8**	**0.298**

Table 7: Example Haikus from CharRNN

```
CharSyllableRNN(
  (lstm): LSTM(67, 1024, batch_first=True, dropout=0.4)
  (feature): Sequential(
    (0): Linear(in_features=1, out_features=1024, bias=True)
    (1): ReLU()
  )
  (dropout): Dropout(p=0.4, inplace=False)
  (fc): Sequential(
    (0): Linear(in_features=1024, out_features=67, bias=True)
  )
)
```

Figure 4: Char RNN Architecture + Syllable Structure

1. For each line of the poem - we perform a character-level generation of the line.

2. Along with that, we append the syllable information by passing the number of syllables in that line through a dense layer to form the hidden dimension representation.

3. Lastly, we add this into the hidden state and the cell state of the LSTM network.

4. Extending our explanation of CharRNN, we one hot encode the text line-wise in each poem.

5. In a loop of three, we pass the tensors into our LSTMs. We first add the dense representation of the syllable count according to the line number and we get the output for each line.

6. Hyperparamater Tuning:

   - Number of LSTM layers = 1
   - Number of epochs = 30
   - Dropout = 0.4
   - Learning rate = 0.0001
   - Batch size = 256
   - Adam Optimizer
   - Cross Entropy Loss as criterion

3.5.2 Results

1. The preliminary results are about an average GRUEN score of 0.348 with a median of 0.31. This has improved only a little from the CharRNN model. (Table 8)

Mean	0.348
25%	0.2162
50%	0.31
75%	0.4252
Max	0.8921

Table 8: Gruen Score of Char-Extension

2. However, the main motto was to improve the structure of the poem, we can observe that the average number of syllables across line are - Line 1: 4.96, Line 2: 7.56, and Line 3: 6.2.

3. In addition, the median of the distribution line wise is - Line 1: 5, Line 2: 7 and Line 3: 6 - showing the generated data learned the structure of the poems in a better way. (Table 9)

3.5.3 Sample Haikus

Some sample haikus generated can be seen in Table 10

	Line 1	Line 2	Line 3
Mean	4.96	7.56	6.2
25 %	3.00	5.50	4.70
50 %	5.00	7.00	6.00
75 %	9.60	10.00	11.00
Max	25.00	19.00	23.00

Table 9: Syllable Structure Score of Char-Extension

Sample Haiku	Meter	Quality
Spring is a long time Apart, I still miss you, my Days spring is my love.	**5, 7, 5**	**0.3053**
The world is just meat And you are the stomach who Soars like butterflies?	**5, 7, 5**	**0.3361**
I feel a little Bit like a butterflies ass This year is madness.	**5, 7, 5**	**0.343**

Table 10: Example Haikus for CharRNN - Extension

3.6 Generative Pre-trained Transformer

3.6.1 GPT-J [9] with Syllable Patterns

1. We started off by exploring ways to try to fine-tune GPT2 and its variants but due to computational constraints, we realized that this task seemed too ambitious.

2. We then came across the GPT-J Deep Haiku model on the Hugging Face which uses the technique that improves generalization by using the domain information contained in the training data of related tasks.

3. For Deep Haiku[8], GPT-J was fine-tuned by teaching it to perform the following four tasks:
   - Generate a Haiku for a given topic using text;
   - Generate a Haiku for a given topic using phonemes;
   - Translate a Haiku from text to phonemes;
   - Translate a Haiku from phonemes to text;

4. This fine-tuning was done with the help of LoRa [11](Low-Rank Adaptation) which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks.

5. As shown in the results below, there's a way to teach large Transformers about counting syllables using multitask learning. Note that it is not 100% perfect; many generated Haikus do not follow the [5, 7, 5] pattern. But it did dramatically increase the probability of following it. And it seemed to have retained its original language training on many different subjects.

6. For future work, there may be a way to use a custom tokenizer for Transformer models that is based on phonemes and syllables, not word parts. The system would inherently be able to see and replicate the meter of prose. Doing so would eliminate the need for multitask learning. The tricky part will be retaining the knowledge from the initial training that was performed on word parts.

3.6.2 Results

We generated about 1500 poems -

1. GRUEN score statistics The average GRUEN score was 0.544 - with a median

of 0.550. The max GRUEN score achieved was 0.876 for the poems generated. Hence the quality of the poems generated is better than the Haiku generated by all other models. (Table 11)

Mean	0.544
25%	0.420
50%	0.550
75%	0.666
Max	0.876

Table 11: Gruen Score - GPT-J

2. We were able to achieve a good GRUEN score and good syllable structure of the Haikus ,we can observe that the average number of syllables across line are - Line 1: 4.96, Line 2: 7.05, and Line 3: 5.10.

3. The median of the distribution line wise is - Line 1: 5, Line 2: 7 and Line 3: 5 - showing the generated data learned the structure of the Haiku in a better way. (Table 12)

	Line 1	**Line 2**	**Line 3**
Mean	4.96	7.05	5.10
25 %	5.00	7.00	5.00
50 %	5.00	7.00	5.00
75 %	5.00	7.00	5.00
Max	7.00	13.00	16.00

Table 12: Syllable Structure Score - GPT-J

3.6.3 Sample Haikus

Some sample haikus generated can be seen in Table 13

4 Conclusion

As we can see in Table 14 - SpaCy, our simple baseline, even though has a good average syllable structure, the quality of the haikus is pretty low majorly because it concatenates words in the haiku randomly. We see a slight improvement in GRUEN quality using CharRNN (our strong baseline) but since it is a character-based model, it doesn't take into account the syllable structure. To further extend it, we append syllable information to our CharRNN which improves both GRUEN score and mean syllable structure. Lastly, we experiment with GPT based models which without a doubt give us best results in terms of GRUEN score and syllable structure.

Sample Haiku	**Meter**	**Quality**
Be a part of the Ritual of nature in This beautiful park.	**5, 7, 5**	**0.7826**
There is a certain Beauty to nature that I Can't find in people.	**5, 7, 5**	**0.7086**
I was sleeping good And I woke up to the scent Of burning autumn.	**5, 7, 5**	**0.8120**

Table 13: Syllable Structure Score - GPT-J

5 Future Scope

For training our charRNN_Syllable type of structure we used only cross entropy loss, we can add structure based loss with it to ensure constrained structure output to see if it helps. As the next possible steps to improve the performance of GPT-J model is to experiment with data augmentation to improve the quality and quantity of the training set and to implement the constraint of toxicity level in the generated text. Also, another direction that can be used to extend this is to implement style transfer from one haiku to another.

Model	Mean GRUEN	Mean Syllable
SpaCy	0.2928	[4.91,6.78,4.92]
CharRNN	0.3396	[7.58,8.71,8.03]
CharRNN with Syllables	0.348	[4.96,7.56,6.2]
GPT-J	0.544	[4.96,7.05,5.10]

Table 14: Conclusion: Comparison of our models

6 Acknowledgements

We'd like to thank Professor Mark Yatskar for the informative lectures and our Lead TA Shreya Havaldar for constantly guiding and supporting us throughout the course of the project.

7 Bibliography

1. Haiku Generation Using Deep Neural Networks, Xianchao Wu, Momo Klyen, Kazushige Ito, Zhan Chen, Microsoft Development Co., Ltd ,
2. Automatically Generating Rhythmic Verse with Neural Networks, Jack Hopkins et al, Facebook AI Research.
3. Haiku Generation : A Transformer Based Approach With Lots Of Control, Giacomo Miceli.
4. Tempes Libres Haiku: http://www.tempslibres.org/tl/en/dbhk00.html
5. Poet RNN: https://github.com/sballas8/PoetRNN/blob/master/data/haikus.csv
6. Haikuzo: https://github.com/herval/creative_machines/blob/master/haikuzao/src/main/resources/haiku.txt
7. Haiku Dataset From Kaggle: https://www.kaggle.com/datasets/bfbarry/haiku-dataset
8. Deep Haiku: Teaching GPT-J to Compose with Syllable Patterns https://towardsdatascience.com/deep-haiku-teaching-gpt-j-to-compose-with-syllable-patterns-5234bca9701
9. GPT-J, Mesh-Transformer-JAX: Model-Parallel Implementation of Transformer Language Model with JAX (2021)
10. LimGen, J. Wang, et al., There Once Was a Really Bad Poet, It Was Automated but You Didn't Know It (2021)
11. E. Hu, et al., LoRA: Low-rank Adaptation of Large Language Models (2021)

8 Appendix

8.1 Data Cleaning

```
file = open("/content/haiku.txt", "r")
fastpunct = FastPunct()
#kw_model = KeyBERT()
roberta = TransformerDocumentEmbeddings('roberta-base')
kw_model = KeyBERT(model=roberta)


data = file.readlines()
start = 0
count_split=17
while start < len(data):
    final_dataframe = pd.DataFrame()
    end = start + 342
    split_data = data[start:end]
    count_t = 0
    for line in split_data:
        # print(count_split , ":::", count_t)
        count_t +=1

        line = line.replace(" $", "")
        haiku = line.split("/")
        source = "bfbarry"
        # Correcting punctuation
        haiku = fastpunct.punct(haiku)

        # Finding topics
        keywords = kw_model.extract_keywords(" ".join(haiku), keyphrase_ngram_range=(1, 2), stop_words=None)

        counts = []
        line_phn = []
        for l in haiku:
            count = 0
            phn = phonemize(l, language='en-us', backend='festival',
                            with_stress=False, separator=Separator(phone=None,
                            word=' ', syllable="|"), strip=True)
            words = phn.split(" ")
            line_phn.append(phn)
            for w in words:
              syllables = w.split("|")
              count += len(syllables)
            counts.append(count)

        final_dataframe = final_dataframe.append(pd.Series(haiku + [source] + [keywords[0][0]] + counts + line_phn), ignore_index=True)
    final_dataframe.to_csv("/content/drive/MyDrive/CIS530-Project/Data/Cleanup/cleaned-data_" + str(count_split) +".csv", index=False)
    start = end
    count_split +=1
file.close()
```

Figure 5: Data Cleaning

8.2 Evaluation base

```python
def get_phonemes(line,char="|"):
    '''
    Get Phonemes

    Arguments:
        Input:
            line (str) - text for calculating phn
    '''
    try:
        phn = phonemize(line, language='en-us', backend='festival', with_stress=False,
                separator=Separator(phone=None, word=' ', syllable=char), strip=True)
    except:
        # 1 syllable
        phn = ""
    return phn


def syllable_count(sen, char = "|"):
    '''
    Get Phonemes

    Arguments:
        Input:
            sen (str) - phonemized structure split by char
    '''
    return sum([len(ph.split(char)) for ph in sen.split(" ")])


def get_data_syllables(data):
    '''
    Get syllable count for each line of poem

    Arguments:
        data =
    '''
    phenoemes_data = np.apply_along_axis(func1d=get_phonemes,arr=np.array(list(data)), axis=0)
    return [syllable_count(sen) for sen in phenoemes_data]


def gruen_loop(data):
    '''
    GRUEN metric fails to run for large amounts of data - hence the loop
    '''
    gruen_score = []
    index = []
    for i in tqdm(range(0,len(data),10)):
        try:
            gruen_score.extend(get_gruen(data[i:i+10]))
            index.extend(list(range(i,i+10)))
        except:
            pass

    return gruen_score,index
```

Figure 6: Evaluation base Script

8.3 Strong baseline - Extension

```
class CharSyllableRNN(nn.Module):
    def __init__(self, chars, n_hidden=1024, n_layers=2,
                               drop_prob=0, lr=0.001):
        super().__init__()

        self.n_layers = n_layers
        self.n_hidden = n_hidden
        #lstm layer - here number of characters is the embedding size as the input is one hot encoded
        self.lstm = nn.LSTM(len(chars),n_hidden,n_layers,batch_first=True,dropout=drop_prob)
        # Syllable dense layer
        self.feature = nn.Sequential(nn.Linear(1,n_hidden),nn.ReLU())
        #dropout layer
        self.dropout=nn.Dropout(drop_prob)
        #output layer
        self.fc=nn.Sequential(nn.Linear(n_hidden,len(chars)))

    def forward(self, x, feat, first_line, hidden=None):
        '''
        Forward pass through the network.
        These inputs are x, feature(syllable) and whether it's the first line of poem
        '''

        # add feature info to hiiden state and cell state
        op = self.feature(feat.reshape(-1,1).float())

        if first_line:
            hidden = (op.unsqueeze(0),op.unsqueeze(0))
        else:
            hidden = tuple((torch.add(hidden[0],op.unsqueeze(0)),torch.add(hidden[1],op.unsqueeze(0))))

        out, hidden = self.lstm(x,hidden)
        out = self.dropout(out)
        out = self.fc(out)
        return out,hidden
```

Figure 7: Strong Baseline - CharRNN

8.4 GPT-J architecture

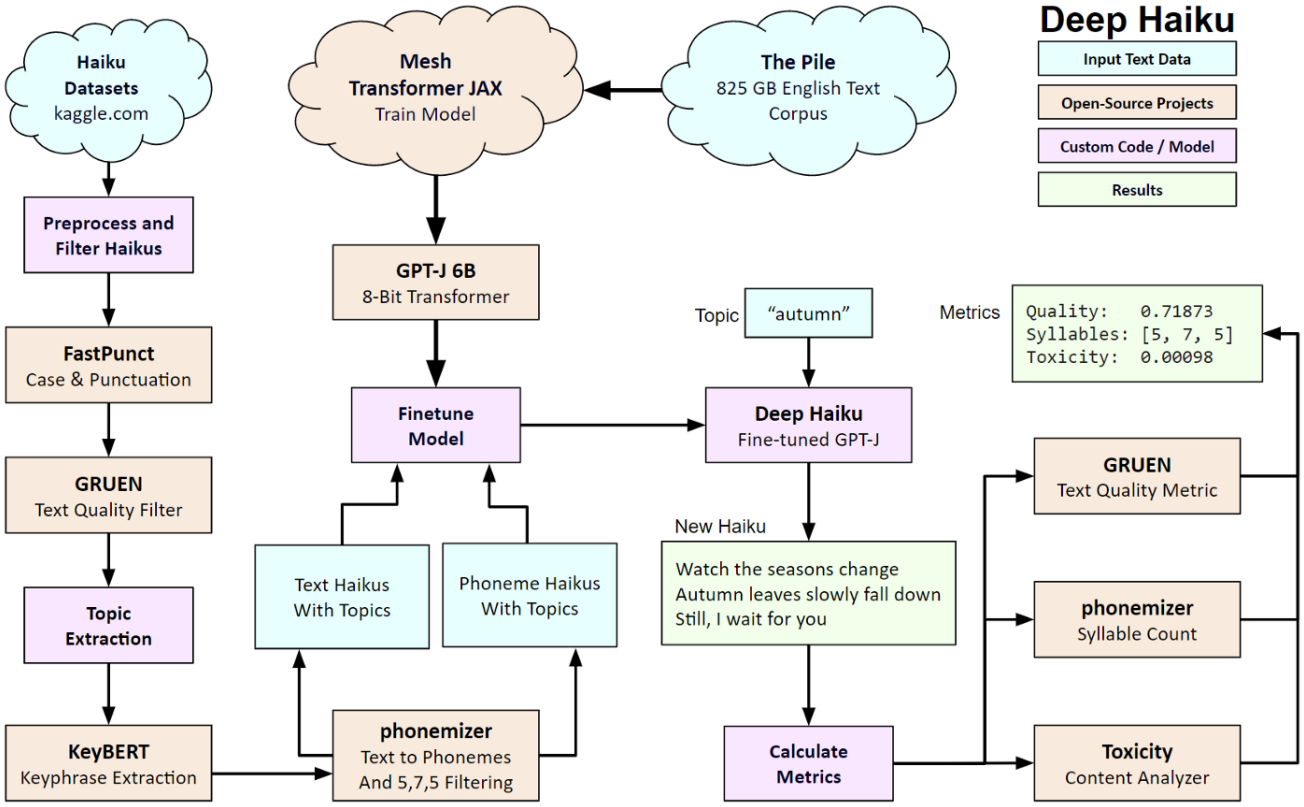

Figure 8: GPT-J architecture