GhostWriter: Dynamic Programming and Deep Learning for Lyric Generation

Stanford CS224N, Custom Project

Niveditha Iyer
Department of Computer Science
Stanford University
nivsiyer@stanford.edu

Tejas Narayanan
Department of Computer Science
Stanford University
tejasn9@stanford.edu

Kiran Bhat
Department of Computer Science
Stanford University
kvbhat@stanford.edu

Abstract

Most songwriters and musicians compose music (rhythm and melody) before writing lyrics to their composition. However, existing music generation models either compose music from a prompt or generate music for lyrics. In our work, we aim to generate lyrics that suit a particular input melody in cadence, mood and meter. We approach this through (i) strictly constrained text generation and (ii) prompted free text generation.

First, we implement a Seq2Seq model to count syllables in English words. Next, we implement and compare eight models that combine algorithmic token prediction and deep learning context-based token generation to varying degrees. Finally, we define lyric evaluation metrics and analyze each model's results. We find that a pre-trained T5 model that is fine-tuned on MIDI notes as well as corresponding lyrics and produces tokens by picking the best branch of a beam search constrained by a syllable count generates lyrics that capture song mood, are syntactically accurate, and align with the metric constraints of the input melody.

1 Key Information to include

- Mentor: Isabel Papadimitriou
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

In recent years, we have seen an increase in the development on ML tools that assist artists in the creative process, from writing assistants to website design chatbots. While text and image generation have been subjects of rich investigation, music generation is a relatively new field. Unlike text and image, music generation involves two modalities - text (lyrics) and sound (melody). Lyrics need to meet a meter as set by the melody and match the mood of the song (both with notes and tempo). Existing work has investigated rhythmic verse generation without melody constraints (Hopkins and Kiela, 2017) and instrumental music without lyrics (Payne, 2019). However, interviews with songwriters demonstrated a gap in music-generation tools - no existing tools generate lyrics to fit pre-existing instrumental music.

2.1 Songwriting Process

We began by interviewing songwriters (J. Segal, A. Krishnan, and K. Bhat as interviewed by N. Iyer) and composers to understand the process of lyric wiring and to quantify metrics we would use while evaluating generated lyrics.

We observed that the songwriting process either begins with (i) a melody that a lyric is then written for (ii) a lyric/theme that a melody is crafted around, with (i) occurring most frequently. Existing music models can be grouped into open-ended instrumental music generation or melody generations from a lyric prompt.

Counterintuitively, there are no models that take in a melody and output suitable lyrics. From our interviews, we discovered three use cases of such a model (i) suggesting a complete verse of potential lyrics, (ii) reorganizing phrases to fit a tune, (iii) forming a lyric around a phrase that a songwriter wanted to use for a verse. The third use case is a generalization of the second use case and both can be incorporated into the first use case. Therefore, we aim to construct a model that will take a melody note sequence as an input and produce a corresponding complete verse.

2.2 Open-Ended Lyric Generation

Lyrics generation is challenging because like open-ended text generation, it lacks a clear objective or evaluation metric. Moreover, we would expect a lyric generation model to have a flair for creativity, with the ability to produce a diverse range of outputs. Generating coherent and meaningful text however is not sufficient, the model must be able to handle ambiguity and produce original responses that capture the nuances of song lyrics.

2.3 Symbolic Representation of Music

In this paper, we focus on the use of symbolic music (rather than audio) as an input to generate lyrics. The main format of symbolic music we work with is called MIDI. MIDI files can store multiple tracks (1 track per instrument typically), and each track contains a sequence of notes. The notes themselves are represented with numbers such as the pitch, the starting time, the ending time, and velocity. See figure 2 for a mapping of pitches to MIDI numbers.

2.4 Lyric Quality Evaluation

The songwriters we interviewed indicated that lyrics could be generally graded on quality based on the following criteria:

1. Melody mood match: Our intuition was that MIDI notes would be correlated to the mood of a song. While our interviewees agreed that notes could be indicative of mood, the tempo of a song is even more indicative of the mood of a song.
2. Melody alignment: Lyrics should meet the syllabic constraints imposed by the length of the sample and the number of notes in a verse.
3. Repetition: While some songs do have entirely non-repetitive lyrics, lyrics with repetition make for memorable, catchy tunes.
4. Semantic coherence: Do the lyrics sensibly progress from one verse to another?
5. Syntactic accuracy: Lyrics do allow for some creative freedom but we would like to preserve conventional English word ordering in most generated verse
6. Room for interpretation: Lyrics with clear-cut meanings read like prose. Interesting and relatable lyrics leave a lot of room for interpretation by the listener.

We decided to evaluate our models on melody alignment, semantic consistency, and syntactic accuracy since these metrics were simpler to quantify. Without extensive human evaluation, we found that we do not have a reliable measure of room for interpretation. Additionally, since we were unable to find consensus from our interviews on the extent of repetition we decided to omit this metric. Finally, we used the melody as an input to our models in order to intrinsically capture melody mood match.

2.5 Our work

In our work, we use a hybrid of algorithmic approaches to generate lyrics that meet syllabic constraints, as well as transformer models to generate semantically and syntactically accurate lyrics. Each of our models falls into the category of either **constrained text generation** or **lyric generation as token prediction**.

Each of our models receives a subset of the following input features - (i) Melody notes of instrumental music, (ii) Time stamps of each note (start, end), (iii) Total syllable count in the ground-truth verse and (iv) Masked lyric inputs (meeting use case (ii) and (iii) as described in Section 2.1).

We find that across our evaluation metrics, the best-performing model was a fine-tuned transformer model that uses syllable-count-constrained beam search to produce tokens.

3 Related Work

To our knowledge, there is no explicit prior work for generating lyrics given music. However, research has been conducted in many similar areas, such as the generation of music from a description (text or note sequence), the generation of music from lyrics, and the constrained generation of poetry.

von Rütte et al. (2022) focused on the generation of music using an "expert" description and a "learned" description. The expert description is a low-fidelity, human-interpretable prompt containing info such as note pitches, note durations, time signature etc. The learned description is higher-fidelity but difficult to interpret. It is created by using a variational auto-encoder framework (VQ-VAE) to generate a latent representation of the music. To generate the music, they feed these descriptions into a Transformer auto-encoder trained on reconstruction. Their model outperformed SOTA models in a variety of metrics, such as controllable generation, modeling capability, and sample quality.

Hopkins and Kiela (2017) demonstrated two generation models for poetry. They first trained a single language model to generate poems with similar style and content to Shakespearean sonnets using a phonetic encoding. While this was successful, they were limited by their training set due to the small number of sonnets compared to the vast corpus of poetry. Considering this, their second approach was to create a character-level poetry generator trained on a large poetry corpus to align the content to that of all poems, as well as a discriminator model trained on the sonnet dataset to fit generated poems to the form of any type of poem. This method proved to be more successful, and they were able to generate poems with varying content styles and forms by changing the datasets the generator and discriminator were trained on. We adopt a similar approach for syllabic lyric generation with MIDI2Seq.

4 Approach

The input to our models is a MIDI file that has been parsed to extract a sequence of melody notes, start times, and end times. We then consider two approaches by which to output a sequence of text tokens (predicted lyrics), namely constrained text generation and seq2seq token prediction (Figure 3).

4.1 Constrained Text Generation

We can consider generating lyrics as a form of constrained text generation where we require the lyrics to meet a syllable count as well as stay aligned with the meter of the song.

4.1.1 Naive Knapsack

We first implemented a naive dynamic programming algorithm to establish a baseline on meeting syllabic constraints. As a proof of concept, we began by taking an algorithmic approach: using the knapsack algorithm to select words from a vocabulary such that the counts of the syllables of the generated words sum to a preset constant. Our syllable-counting process used the CMU Phonetics Dictionary (CMU) and a separate seq2seq model that we implemented and trained to predict the number of syllables in each word (Figure 4). While this approach did produce outputs meeting the constraint, they were not coherent.

4.1.2 Constrained AutoSeq2SeqLM

Our next approach was to use Constrained Beam Search. The general beam search setting involves finding the top-k most probable next tokens at each branch and appending them to the branch, giving us several new branches for consideration. These branches are then pruned to select the top-k branches. In this constrained setting, we append the tokens that will take us closer to fulfilling our syllabic constraints. We constructed an AutoModelForSeq2SeqLM from HuggingFace (Wolf et al., 2020), pretrained on "t5-base", and wrote a CountConstraint class (an instance of transformers.Constraint) that would restrict the number of syllables that the model could produce. The class implements a method to suggest a token that the model uses as the next prediction, a method to check if the constraint has been met, and a method to determine whether the model should predict the suggested token. The CountConstraint class is fed as a parameter to the AutoSeq2SeqLM during initialization. We use a cross-entropy loss between the generated tokens and the true tokens as our training metric for all our models.

4.2 Lyric Generation as Token Prediction

The results of our constrained models (Section 5.4) led us to investigate whether constraint-free generation would produce higher-quality lyrics (semantically and syntactically). We switched to directly using the notes and lyrics instead of purely imposing constraints derived from them. Note that syllable counts would still be implicit in the number of notes in the input MIDI sequence. To this end, we implemented MIDI-BERT, MIDI2Seq, and MIDI-T5, three new lyric generation models. Our intuition is that these models would effectively be "translating" between the MIDI music space and English lyric space.

4.2.1 MIDI-BERT

MIDI-BERT is an original model that uses a MIDI embedding layer composed of a linear layer (256 $\times$ 256), a ReLU layer, and another linear layer (256 x 256). The notes of the song are fed into the MIDI embedding layer to obtain embedding H. We further condition the generation of the lyrics on the lyrics that have already been written for the notes. Inspired by the BERT (Devlin et al., 2019) training process, we mask out some words from lyrics L and then tokenize the masked lyrics giving us a lyric representation X. The tokenized lyrics are concatenated to H, giving us $\{H,\ X\}$. The concatenated embeddings are fed into a BERT model which generates predictions for each of the masked-out tokens.

Since our ultimate goals was to input notes and output lyrics, we decided to look into encoder-decoder architectures such as the Seq2Seq and pre-trained models such as T5.

4.2.2 MIDI2Seq

The Seq2Seq model architecture consists an encoder that processes the input sequence and produces a fixed-size representation that captures its semantic meaning. The decoder then generates the output sequence based on the encoder's representation. The model is trained end-to-end using maximum likelihood estimation, where the goal is to minimize the difference between the predicted sequence and the true sequence.

MIDI2Seq is a Seq2Seq model implemented from scratch that acts like a translation model, between notes and English. It consists of an encoder with three LSTM layers and a decoder with 2 LSTM layers and a TimeDistributed Dense layer. MIDI notes and their timestamps make up the source sequence while tokenized ground-truth lyrics make up the target sequence. Notably, we use "sub-words" as available in the LakhMIDI dataset to train the model instead of full words. While we expect a decrease in syntactic and semantic performance, we tried this approach to test how well the model's generations would align with the time stamps of the input sequence.

4.2.3 MIDI-T5

Since our ultimate goals was to input notes and output lyrics, we needed to look into large encoder-decoder models such as T5. The T5 model (Raffel et al., 2020) is a transformer-based neural network that includes an encoder and decoder with multiple transformer layers. During training, T5 uses unsupervised and supervised learning objectives, including masked language modeling and sequence-to-sequence tasks. The pre-trained model can be fine-tuned on specific downstream tasks and achieves SOTA performance on several NLP benchmarks.

We run three variations of a T5-inspired model. MIDI-T5-Open-Ended is a pre-trained model (using t5-small) that is fine-tuned on the lyrics of the LakhMIDI dataset. MIDI-T5-Custom-NoEmbed is a modified T5 model. We change the size of the Encoder embedding layer to handle an input vocabulary of size 100. This vocabulary comprises of the numbers 0 to 100, representing the full range of MIDI notes observed in our dataset. In addition, we initialize the MIDI embedding layer with randomly sampled embeddings from the decoder layer of the pretrained T5 model. We then discard the embedding layer and use the remainder of the model to generate lyrics. Our intuition was that this would still capture the "music space" and "lyric space" in the encoder and decoder models. Finally, MIDI-T5-Custom is a similarly modified T5 model with an input vocabulary size of 100, where we train and retain all layers including the encoder embedding layer.

4.2.4 Knapsack+GPT

We also explored leveraging pre-trained language model outputs with dynamic programming. The Knapsack-GPT model is a modified knapsack algorithm that takes in a vocabulary of the 200 most-used English words and selects the best-possible ordering to optimize for the predicted probability of the word in the context of the sentence, as calculated by GPT-2 (Radford et al., 2019), while exactly matching the syllable count of the lyrics.

For a given MIDI-lyrics pair, we calculate the best-possible k-syllable sentence for all $k = 1, \ldots, n$, where n is the number of syllables in the true lyrics. For each input to GPT-2, we join the MIDI notes with the currently-considered sentence, separated by an end-of-sequence token. This enables GPT-2 to consider the MIDI notes as part of the context of the sentence, therefore tuning the current output to the song.

5 Experiments

5.1 Data

5.1.1 Scraping YouTube

We began by writing a script to download YouTube videos with their timed-stamped captions. We downloaded a large set of "Official Audio" music videos, filtered to have closed captions. We also wrote a script to download and clean up the closed captions of these videos, which provided us with a dataset of lyrics and their associated timestamps.

Next, we needed to separate the melody from the background music in each music video. We found the tool vocalremover.org, which extracted vocals from the background music. We wrote a Selenium script to ping the website with automated requests. Unfortunately, our IP address was blocked by the system after a single request, likely because it detected that the request was made automatically. We pivoted to the open-source model Omnizart which runs the open-source vocal splitter model Spleeter locally to obtain the isolated vocals of each of our music clips. However, we found that the inaccuracies of the Omnizart vocal transcriptions made this approach infeasible.

5.1.2 The LakhMIDI Dataset

Finally, we came across the LakhMIDI dataset which pairs lyrics with timestamps for $\approx$ 45,000 MIDI files, which meant we did not need to source this data ourselves. We used PrettyMIDI to extract notes and lyrics from MIDI files. However, the MIDI did not identify a distinct melody track, but rather provided all of the separate instrument tracks. To determine which track corresponded to the melody, we calculated a similarity score between each of the tracks and the lyrics, comparing their note onset times with the start time of each syllable in the lyric track. Then, we chose the track with the highest similarity score as the melody track (Figure 5). The data is stored in a CSV containing the melody and lyrics along with time stamps for use in training, validation, and testing. For future data generation, we are able to modify the number of tracks we use or combine the tracks, which might be particularly useful for mood selection.

Further challenges included missing/garbled lyrics and filtering non-English songs. Another challenge arises from the way lyrics are represented in the MIDI file. Rather than a series of words, they are represented as a sequence of sub-words. This makes generating text tricky since we could have a word like “yesterday” or “because” be split into [“yes”, “ter”, “day”] and [“be”, “cause”] respectively. During generation, we need to ensure that “ter” does not appear out of order or with another word’s syllables such as in “yestercause”. To remove this complexity, we concatenate these sub-words to form full words before feeding them to all models except MIDI2Seq.

5.2 Evaluation method

We decided to evaluate the performance of our lyric generation models based on syllable alignment, linguistic acceptability, and sentiment capture.

5.2.1 Syllable Alignment

We define syllable alignment, a metric that describes how well the syllables of the generated lyrics line up with the input note sequence. To evaluate the syllable alignment of our generated lyrics, we computed the mean alignment error (MAE) as follows:

$$\text{MAE} = \frac{\sum |\text{syllable count of actual lyrics} - \text{syllable count of generated lyrics}|}{\text{total number of songs}}.$$

The MAE describes the average syllable count disparity between our generated lyrics and the actual lyrics for the same note sequences. Consequently, we want our MAE to be close to 0.

5.2.2 Linguistic Acceptability

Linguistic acceptability measures how grammatically acceptable a sentence or phrase is. To evaluate the linguistic acceptability of our generated lyrics, we used a distilled version of the BERT base model that was finetuned to classify text as either "acceptable" or "unacceptable". It was finetuned on the Corpus of Linguistic Acceptability (CoLA), a dataset containing sentences and their labels, "acceptable or "unacceptable".

While conducting initial experiments, we found that the model was too lenient on acceptability, and classified some unacceptable sentences as acceptable. To address this, we used the score (output of the softmax for a particular class) of the prediction. The closer the score is to 1, the more confident the model is about its prediction for a particular class.

We then computed a linguistic acceptability rate (LAR) as follows:

$$\text{LAR} = \frac{\sum 1\{\text{generated lyrics are classified as "acceptable" with score} > 0.8\}}{\text{total number of songs}}$$

The LAR describes the rate at which our lyric generation model produces linguistically acceptable lyrics. For example, if the LAR = 0.42 for a dataset of 1000 generated lyric outputs, then 42% of the generated lyrics are linguistically acceptable. We want the LAR to be close to 1.

5.2.3 Sentiment Capture

Sentiment capture is a measure of how well our generated lyrics capture the sentiment of the input note sequence. To get the sentiment of our lyrics, we used a distilled version of the BERT base model that was fine-tuned to classify text as either "positive" or "negative". It was fine-tuned on the Stanford Sentiment Treebank, a dataset containing opinions and their sentiment ("positive" or "negative").

We then computed a same sentiment rate (SSR) as follows:

$$\text{SSR} = \frac{\sum 1\{\text{sentiment of generated lyrics = sentiment of actual lyrics}\}}{\text{total number of songs}}.$$

The SSR describes the rate at which our generated lyrics have the same sentiment as the actual lyrics for the same note sequences. For example, if the SSR = 0.61 for a dataset of 1000 songs and their respective generated lyrics, it means that the generated lyrics had the same sentiment as the actual lyrics 61% of the time. We want a model's SSR to be above 0.5 (which indicates the generated lyrics have a random sentiment).

5.3 Experimental details

The models had varying levels of computational complexity and therefore required different training environments. Training and evaluation for the Knapsack Generation, AutoModelforSeq2SeqLM, and MIDI2Seq were done on Google Colab. Training and evaluation for MIDI-BERT, MIDI-T5, and Knapsack+GPT were done on a g5.2xlarge machine on AWS.

All deep learning models described in this paper were trained with the following hyperparameters:

Parameter	**Value**
Learning rate	10^{-4}
Batch size	16
Training epochs	300

5.4 Results and Analysis

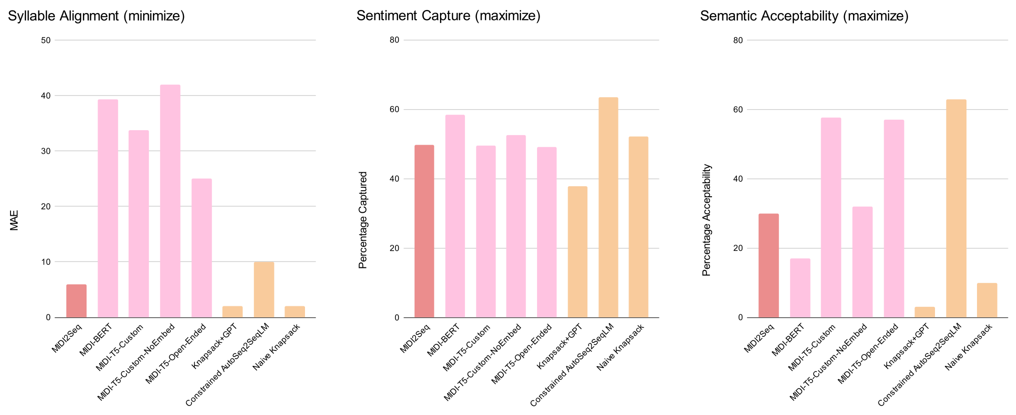


Figure 1: Model performance on evaluation metrics. Models in orange were explicitly given the syllable count constraint. Models in pink were not given the syllable count but only the input notes. Models in dark pink were given input notes and time stamps.

We were surprised to see the consistently poor performance on Sentiment Capture (essentially random performance). We hypothesize that like our interviews with songwriters suggested, this was because MIDI notes do not capture song mood. The patterns observed in syllable alignment and semantic acceptability are analysed below. Lyric outputs of various models are shown in Figure 6.

The Naive knapsack is purely algorithmic and produces lyrics with a low MAE of 2 syllables. However, as we expected it performs very poorly on semantic acceptability (10%) and sentiment capture (52%). It does not reach an MAE of 0 since we use a random sampling of words and we place a restriction on how many iterations the algorithm can make through the dictionary of words.

The MIDI2Seq model takes time stamps of notes as an input along with notes and generates sub-words consisting of 1 or 2 syllables. We hypothesize that the inability to generate long tokens might be the cause for its low syllable alignment MAE (6 syllables). In addition, the limited data did not allow the model to learn how to join sub-words into words, resulting in a low semantic acceptability rate.

MIDI-BERT was unable to generate meaningful outputs, likely because there was not enough data to train the MIDI embedding layer, as well as the fact that BERT is not suited for free text-generation as well as other models, such as T5, are.

The Constrained AutoSeq2SeqLM Model generates lyrics with the highest Semantic Acceptability across models while achieving a low syllable alignment MAE. This is likely because we finetune "t5-base" and force the model to produce lyrics constrained by a beam search that does not allow the model to exceed the syllable count provided in our CountConstraint class. This model most effectively combined an algorithmic approach with a deep learning model to generate high-quality lyrics. A visual inspection however reveals that the generated lyrics read like written prose and leave little room for subjectivity. This might be because the beam search preferentially selects branches with shorter words, creating simpler sentences.

Knapsack+GPT was tied for the lowest MAE across all models, a result of its algorithmic requirement to meet syllable count. However, it performed very poorly on the LAR and SSR metrics. We believe this is because it was restricted to only using the most-common English words due to model evaluation speed, so it was unable to create comprehensible outputs. Moreover, the knapsack problem's inherent assumption is that the best possible sentence of length n would be equal to the best possible sentence of length $n - k$ followed by a k-syllable word. This is a strong assumption to make, as the choice for the previous words would significantly impact the best possible choice for the last word. As a result, despite its dynamic-programming formulation, this approach is still "greedy", in the sense that choices about optimality made earlier on may affect future choices in a sub-optimal way.

Upon manually inspecting our output, we noticed that the T5 model was occasionally generating lyrics that sounded familiar. We discovered the T5-Custom-NoEmbed model had memorized certain phrases that we found in the input data. We found instances of "having the time of your life" which traced back to the input data as coming from ABBA's Dancing Queen and "walk a lonely street" from Green Day's Boulevard of Broken Dreams. This suggests that we had overfit our model or not trained on enough data.

MIDI-T5-Open-Ended generated one output we particularly liked. We recorded ourselves singing the unedited output on top of the track it was generated from. This recording, titled "Do You Remem", can be viewed here. The MIDI input to the model and produced lyrics are shown in Figure 7.

6 Ethical Considerations

Open-ended lyric generation models raise ethical considerations that we take seriously. In this section, we discuss the ethical considerations that were taken into account during the development and deployment of our lyric generation model.

To mitigate concerns about biased lyric generation, we selected a diverse set of training data across genres and artists. Since text generation models have the potential to generate offensive content, we retained only English lyrics so that we could understand the content being generated. We foresee implementing a filter against offensive words and phrases. We would like to flag generated lyrics that might be similar to existing songs to allay concerns regarding ownership of the generated content.

We recognize the potential impact of our work on professional lyric writers. Our interviews indicated that songwriters take pride in their work and would likely use such models as writing assistants - to provide inspiration, or re-arrange lyrics to fit a melody. This could also boost accessibility to the creation of music just as models like DALL·E have already helped independent songwriters generate cover art without commissions.

7 Conclusion

In this paper, we demonstrate algorithmic and deep learning approaches to lyric generation from input MIDI sequences. We implement a syllable-count model, one purely algorithmic lyric-generation method, and six deep learning methods lyric-generation methods that leverage dynamic programming. We demonstrate that algorithmic approaches with strict constraints on syllable counts struggle to produce semantically meaningful and syntactically accurate lyrics. Syllable constraint-free transformer models generated higher-quality input melody sequences. We find that the best model in terms of quantitative results is Constrained AutoSeq2SeqLM, while the model with the best qualitative results is MIDI-T5-Open-Ended.

References

CMU. The cmu pronouncing dictionary.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. ArXiv:1810.04805 [cs].

Ryan Epp. Predicting english pronunciations.

Jack Hopkins and Douwe Kiela. 2017. Automatically generating rhythmic verse with neural networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, page 168–178, Vancouver, Canada. Association for Computational Linguistics.

Christine Payne. 2019. [link].

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. ArXiv:1910.10683 [cs, stat].

Dimitri von Rütte, Luca Biggio, Yannic Kilcher, and Thomas Hofmann. 2022. Figaro: Generating symbolic music with fine-grained artistic control.

Mickaël Tits. 2015. Master thesis mickael tits 2014 - development of an optical motion capture setup for feature extraction and statistical analysis of the pianist's expert gestures.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Huggingface's transformers: State-of-the-art natural language processing. ArXiv:1910.03771 [cs].

A Appendix

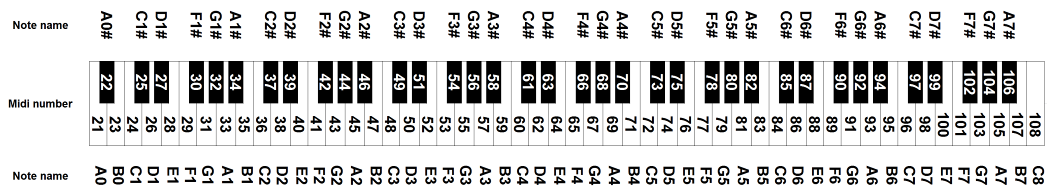


Figure 2: A mapping of pitches to their respective MIDI representation (Tits, 2015).

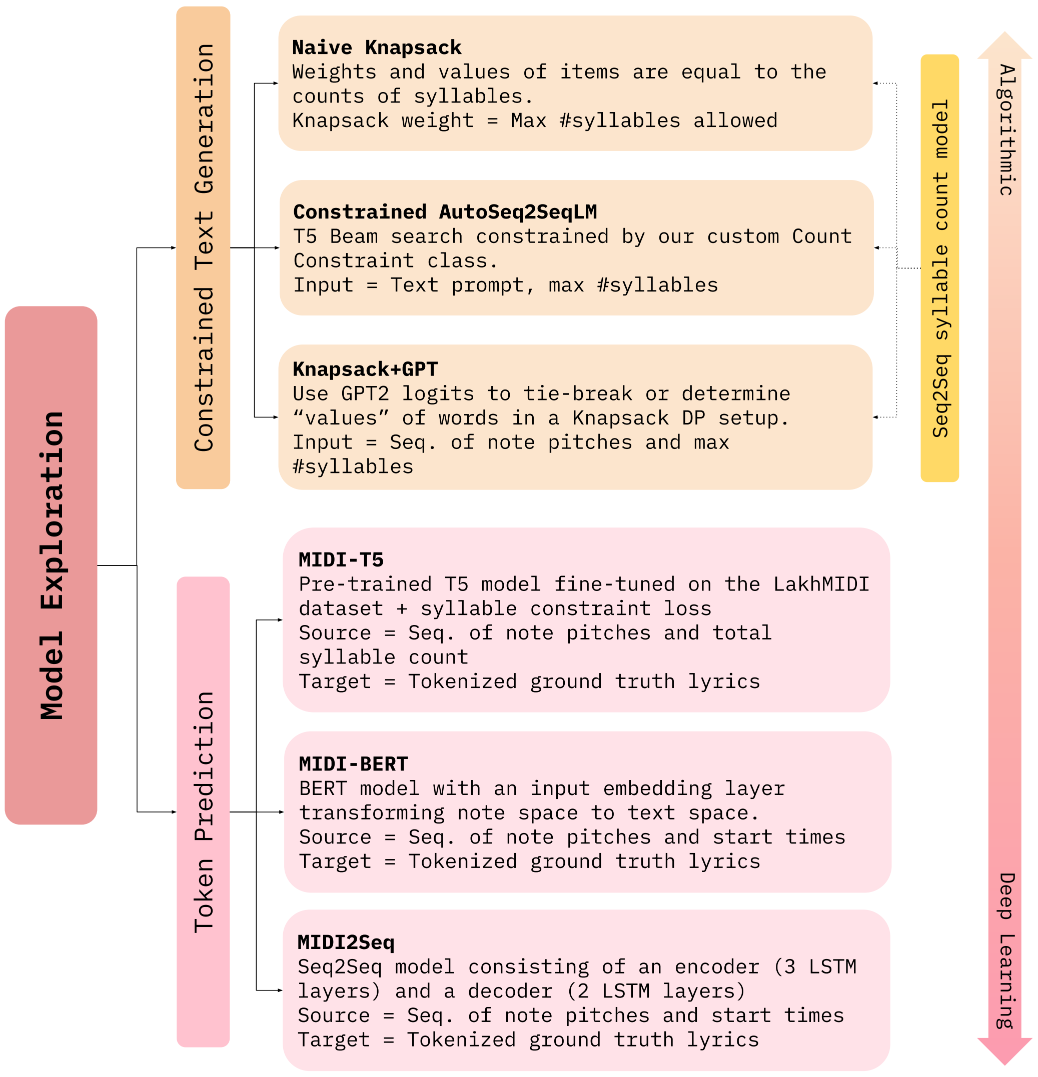

Model Exploration
Constrained Text Generation
Naive Knapsack
Weights and values of items are equal to the counts of syllables.
Knapsack weight = Max #syllables allowed
Constrained AutoSeq2SeqLM
T5 Beam search constrained by our custom Count Constraint class.
Input = Text prompt, max #syllables
Knapsack+GPT
Use GPT2 logits to tie-break or determine "values" of words in a Knapsack DP setup.
Input = Seq. of note pitches and max #syllables
Token Prediction
MIDI-T5
Pre-trained T5 model fine-tuned on the LakhMIDI dataset + syllable constraint loss
Source = Seq. of note pitches and total syllable count
Target = Tokenized ground truth lyrics
MIDI-BERT
BERT model with an input embedding layer transforming note space to text space.
Source = Seq. of note pitches and start times
Target = Tokenized ground truth lyrics
MIDI2Seq
Seq2Seq model consisting of an encoder (3 LSTM layers) and a decoder (2 LSTM layers)
Source = Seq. of note pitches and start times
Target = Tokenized ground truth lyrics
Seq2Seq syllable count model
Algorithmic
Deep Learning

Figure 3: Model Architectures we explored for Lyric Generation from MIDI files

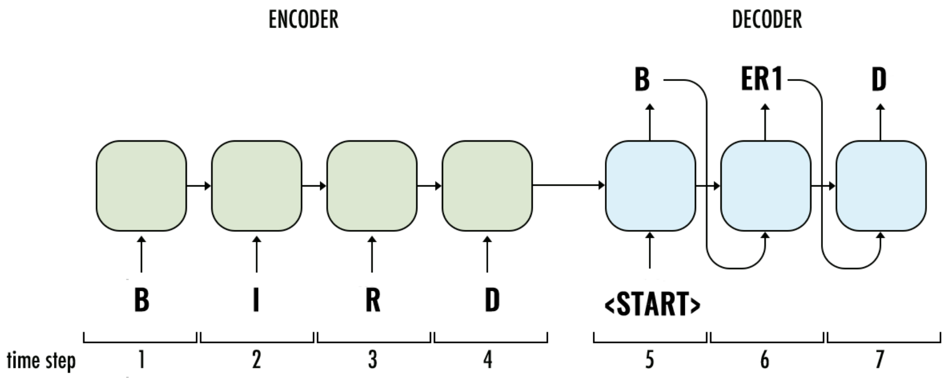


Figure 4: Seq2Seq model for phoneme breakdown prediction. We obtain the number of syllables by counting the number of phonemes with stress markers. Achieves 98.1% accuracy in syllable counting. (Epp)

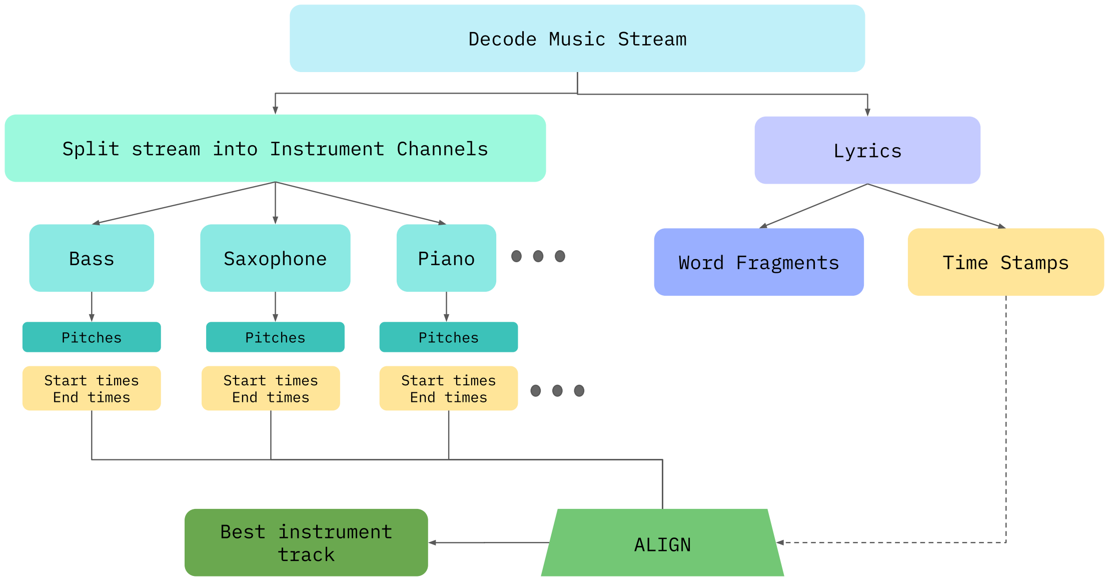
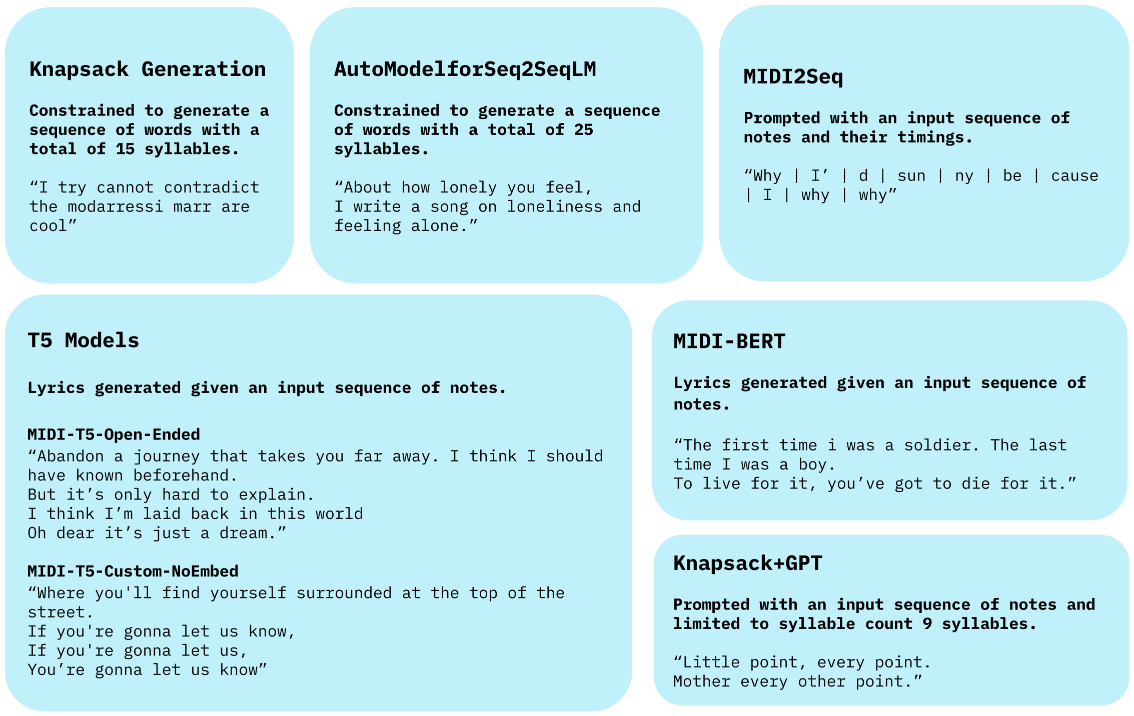


Figure 5: Process by which dataset was collected and processed. We first obtain the note and lyric data with PrettyMIDI, then use our alignment algorithm to find the instrument track that best aligns with the lyric timing. We use these best tracks and their corresponding lyrics as the melody-lyric pairings in the dataset.



Knapsack Generation

Constrained to generate a sequence of words with a total of 15 syllables.

"I try cannot contradict the modarressi marr are cool"

AutoModelforSeq2SeqLM

Constrained to generate a sequence of words with a total of 25 syllables.

"About how lonely you feel,
I write a song on loneliness and feeling alone."

MIDI2Seq

Prompted with an input sequence of notes and their timings.

"Why | I' | d | sun | ny | be | cause | I | why | why"

T5 Models

Lyrics generated given an input sequence of notes.

MIDI-T5-Open-Ended
"Abandon a journey that takes you far away. I think I should have known beforehand.
But it's only hard to explain.
I think I'm laid back in this world
Oh dear it's just a dream."

MIDI-T5-Custom-NoEmbed
"Where you'll find yourself surrounded at the top of the street.
If you're gonna let us know,
If you're gonna let us,
You're gonna let us know"

MIDI-BERT

Lyrics generated given an input sequence of notes.

"The first time i was a soldier. The last time I was a boy.
To live for it, you've got to die for it."

Knapsack+GPT

Prompted with an input sequence of notes and limited to syllable count 9 syllables.

"Little point, every point.
Mother every other point."

Figure 6: Lyric outputs of various models.

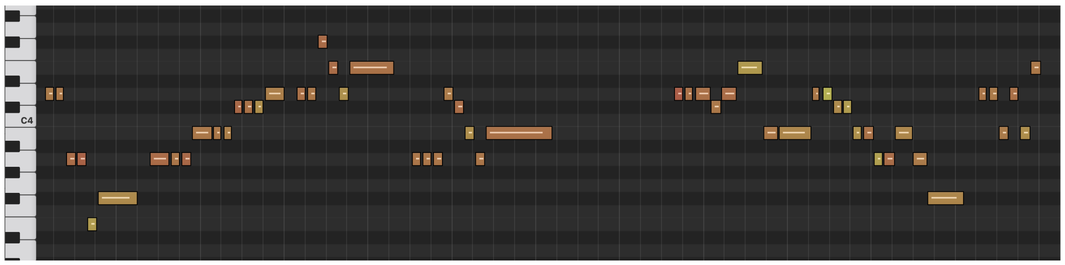


Figure 7: MIDI input for "Do You Remem". The lyrics, produced by MIDI-T5-Open-Ended, are as follows: "GOES THIS WAY. / MY HEART GETS BLOWN AWAY / THESE EYES ARE BORN AGAIN. / DO YOU REMEMBER HOW WE WERE? DO YOU REMEM"