# Structured Text Generation for Spanish Freestyle Battles using Neural Networks

Pedro Dal Bianco
*Instituto de Investigación en Informática LIDI*
*Becario Doctoral UNLP*
*Universidad Nacional de La Plata*
Buenos Aires, Argentina
pdalbianco@lidi.info.unlp.edu.ar

Iván Mindlin, Laura Lanzarini,
*Instituto de Investigación en Informática LIDI*
*Universidad Nacional de La Plata*
Buenos Aires, Argentina
laural@lidi.info.unlp.edu.ar, mindlinster@gmail.com

Franco Ronchetti, Waldo Hasperué
*Instituto de Investigación en Informática LIDI*
*Universidad Nacional de La Plata*
*Comisión de Investigaciones Científicas de la Pcia. de Bs. As. (CIC-PBA)*
Buenos Aires, Argentina
fronchetti, whasperue@lidi.info.unlp.edu.ar

Facundo Quiroga
*Instituto de Investigación en Informática LIDI*
*Becario Postdoctoral UNLP*
*Universidad Nacional de La Plata*
Buenos Aires, Argentina
fquiroga@lidi.info.unlp.edu.ar

2021 XLVII Latin American Computing Conference (CLEI) | 978-1-6654-9503-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/CLEI53233.2021.9639929

***Abstract*—As the presence of artificial intelligence has increased in a variety of different areas, the use of machine learning and deep learning techniques for creative purposes has also risen significantly in recent years. Works of this kind within the area of natural language processing (NLP) are typically neural models used for fiction or lyrics generation. Those works are in most cases in English and adapting them to other languages is not feasible.**

**In this work, we develop a Spanish text generator system for the rap sub-genre known as freestyle. Freestyle songs present unique challenges for text generation given that performers compete with one another in a lyric improvisation contest.**

**Given the low availability of freestyle text, especially in Spanish, we collected two separate datasets, one with freestyle lyrics and the other, larger, with rap lyrics, which are more readily available. The rap dataset can be used for pretraining, and the freestyle dataset for finetuning on the generation task. Furthermore, we design a neural network-based generation model that takes into account both the structure of freestyle and the low data availability. The model was able to generate realistic freestyle verses in Spanish.**

***Index Terms*—Natural Language Processing, Natural Language Generation, Neural Networks, Recurrent Neural Networks, Lyrics Generation, Spanish, Freestyle**

## I. Introduction

Natural Language Processing and more specifically Natural Language Generation is one of the main areas of interest within Artificial Intelligence. In the beginning, the techniques used templates to generate text. However, the emergence of deep learning techniques applied to this particular field has allowed a great advance in the current state of the art [1] [2]. This approach uses a neural language model that given a sequence of words is capable of generating a probability distribution for the next word [3]. Nowadays, the availability of large text databases through the internet allows these neural language models to be very robust [4]. A recent model, GPT-3 [5], is a clear example of this phenomenon. It was trained on a version of the Common Crawl database [6], which consists of around a trillion words. GPT-3 achieves very good results for text generation, via training first on large databases such as the one mentioned and then performing fine-tuning on specific texts that encode a subtask. For example, [7] employs this approach for the generation of different particular genres such as poetry, dialogues, or humorous texts.

Particularly, the generation of artistic structured texts, such as poetry or lyric, among others, requires a need for flexibility and structure prediction that can be fulfilled by using deep learning techniques with specific rules about metric, accentuation, and rhyme of the generated text. For instance, [8] combines LSTM networks for the generation of poetic text with finite state machines that accept text that meets the desired structure. [9] also uses LSTMs for the generation of several candidate lines to create sonnets and the selection of the one that best suits the desired structure based on the metric and rhyme.

It is possible to find works that deal with the generation of poetic text in languages other than English, such as [10] where recurrent neural networks trained on classical Chinese poetry are used or [11] that trains a syllable-based model with poems in Italian by Dante Alighieri. However, the only work of this kind found to use the Spanish language is the already mentioned [8]. Although this last work focuses mainly on the generation of text in English, its last section shows how poetry in Spanish can also be generated using the developed model trained with a bank of lyrics of songs in Spanish and text extracted from Wikipedia.

978-1-5386-5541-2/18/$31.00 ©2018 IEEE

No works were found at the time of this writing that deals with the generation of texts for freestyle battles. The closest work we found is [12], where a neural language model is used to generate English rap lyrics.

### A. Freestyle battles

Freestyle Battles are a subgenre of rap, where performers compete against each other by measuring who is capable of improvising the best verses [13]. In recent years this genre has grown in popularity. Spanish freestyle competitions are held both locally and internationally as massive events. In these battles, participants' contributions are scored over six rounds. In each round, they have to improvise under some particular rules and a jury will evaluate each stanza with a score from 0 to 4 points depending on the quality of the rhyme and how well it follows the rules of that particular round.

Text generation for freestyle battles in Spanish has some particular difficulties that are not present in other poetic genres. Given its improvised nature, there are almost no databases containing transcripts of these battles and most of the material is only available as video. Also, freestyle battles commonly include many anglicisms (English words or expressions within a Spanish sentence), in addition to deformations of words or slang. This makes it difficult to use transfer learning techniques such as Spanish pre-trained embeddings that might not contain these expressions or English words. Also, when trying to identify rhymes, we must consider that said expressions probably will not fit Spanish accentuation rules. Finally, generating text in Spanish means that most similar works in English, such as trained language models, will not be directly applicable to this task.

Furthermore, the fact that the text has to be improvised in real-time in response to the rhyme of a competitor while respecting appropriate rhyme schemes and metrics makes this genre especially interesting for the development of an automatic generation model.

In this work, we perform the following contributions:

1) We collect a text dataset of Spanish rap songs
2) We also collect a smaller text dataset of freestyle performances, extracted and corrected from video
3) We propose and evaluate a neural network-based generation model, which can generate realistic freestyle songs. Specifically, via the use of reverse generation and bidirectional LSTMs, the model can capture the notion of rhyme and metric.
4) We compare and analyze several tokenization and rhyming models to compensate for the low data availability of freestyle text.

## II. Dataset generation and analysis

Due to freestyle competitions being a relatively recent phenomenon (the first official international competition in Spanish was held in 2005) it is expected that most of the material available today of this genre is in audio or, more commonly, video format. Additionally, the improvisational nature of the genre and the fact that its execution is accompanied by gestures, pronunciation, and interaction with the public complicates obtaining data suitable for machine learning. Although large amounts of this type of material that can be accessed through the internet, very little of it is available in text format such as transcripts, necessary to train language models.

TABLE I
GENERATED DATABASES SUMMARY.

| | Rap DB | Freestyle DB |
|---|---|---|
| Documents | 833 | 36 |
| Lines | 54364 | 8355 |
| Words | 381834 | 68763 |

Given that no freestyle text database exists at the time this work was written and transcriptions could not be found in abundance on the internet, we considered the use of transfer learning. To pretrain the model on similar material that is easier to access in large quantities, we generated a *rap dataset* that contains lyrics of rap songs or similar genres. Through training on this dataset, the model was able to obtain notions not only about the general structure of the Spanish language but also about more specific issues such as the structuring of a text in verses or possibly rhyme. Furthermore, although these do not correspond exactly to transcriptions of freestyle battles, as the genres are similar, they share many aspects such as the vocabulary or the rhymes.

This database was collected by performing web scrapping on lyrics websites over the profiles of the most relevant artists of these genres. It contains 833 different song lyrics, all in Spanish, containing 54,364 lines and a total of 381,834 words.

To finetune the model, we also collected a second *freestyle dataset* with transcripts of battles. The data used to generate this dataset was much more difficult to obtain, and therefore the amount of text it contains is significantly less than the rap database. The only source of Spanish freestyle battle transcripts found was a wiki [14] which has 23 transcripts and several of them are incomplete. These transcripts were completed and corrected manually from videos of the respective battles found on YouTube. Other transcripts obtained also manually transcribing videos of other battles were added to complete the database with a total of 36 transcripts composed of 8355 lines and 68763 words.

A description of both databases can be found in Table I. These are public and can be accessed in this work's repository, alongside all the model and preprocessing code [15].

### A. Vocabulary analysis

In order to perform a vocabulary analysis, two trimmed histograms were generated corresponding to the words in each of the datasets, excluding stop words. These can be seen in the figures 1 and 2 and allow us to observe that the use of English words (such as "flow", "freestyle" or "baby"), deformations of certain words (for example "pa"' is used in replacement of the Spanish word "para") and the inclusion of onomatopoeias (as the case of "yeah" or "ey" among others) are a very common occurrence in both datasets. This must necessarily be taken into account when considering how to generate rhymes:

the phonetics of English words must be analyzed differently from that of Spanish words; and when using transfer learning through pre-trained embeddings, since these might do not consider the tokens corresponding to word deformations or onomatopoeias.

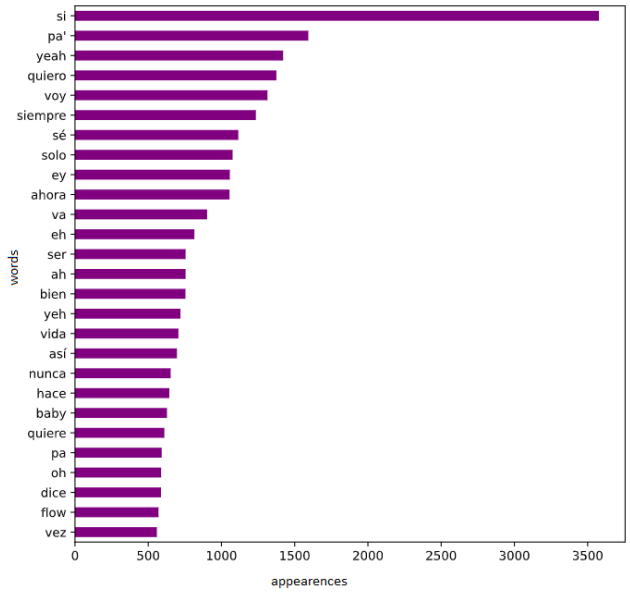


Fig. 1. Most frequent words over Rap DB

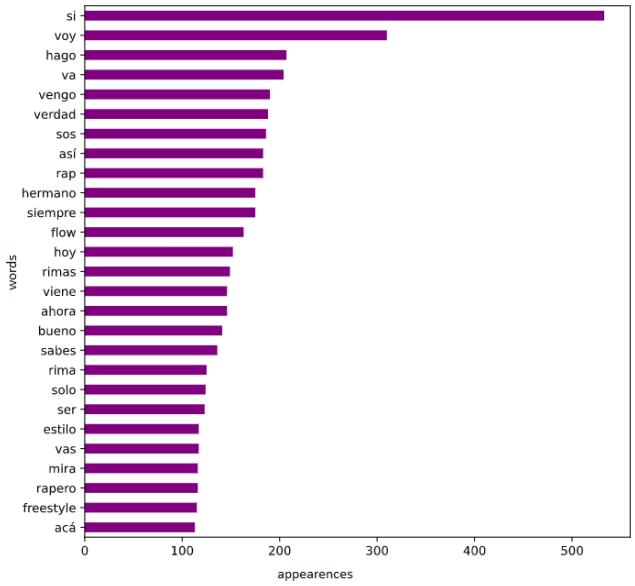


Fig. 2. Most frequent words over Freestyle DB

## B. Tokenization

Three tokenization levels were considered: word, syllable, and Byte Pair Encoding (BPE). In all cases, we add a special token "[SEP]" to indicate the ending of a verse. Character-level tokenization was not taken into account as it is considerably harder to generate verses that rhyme by modeling at the character level than by using words or syllables. Additionally, the vast majority of NLP tasks currently employ word or sub-word tokenization.

*1) Word level tokenization:* Applying word-level tokenization over the rap dataset resulted in 28.424 different tokens. Of these, 12.919 of these appear only once and 19.777 appear at most 3 times, (45.45% and 69.58% respectively). For the freestyle dataset, the problem is even greater. This dataset has 8681 different tokens of which 4723 (54.41 % of the total) appear only once and 6847 of these (78.87 %) appear at most 3 times. Table II summarizes these observations.

TABLE II
SUMMARY OF THE OBSERVATIONS MADE REGARDING THE AMOUNT OF TOKENS WITH LOW FREQUENCIES AT WORD LEVEL.

| Word Tokens | Rap DB | | Freestyle DB | |
|---|---|---|---|---|
| Total tokens | 28424 | 100% | 8681 | 100% |
| Tokens that appear once | 12919 | 45,45% | 4723 | 54,41% |
| Tokens that appear at most 3 times | 19777 | 69,58% | 6847 | 78,87% |

The word distribution of these datasets results in a significant problem since it is likely that the language model cannot obtain useful information about how to represent or use the words represented by these rare tokens. Therefore, a large part of the dataset cannot be exploited. Because of this, tokenization at the word level might not be ideal for this problem with this particular data set.

One of the possible reasons for that many low-frequency tokens is the presence of possible typos or different ways of writing certain expressions or onomatopoeias: In the histogram shown above one can see both the "yeah" and "yeh" tokens, which could be referring to the same expression. Token "pa"' can also be seen replacing the word "para", even the token "pa" (without the final apostrophe) is also present, having the same meaning. This results in the total frequency of the word "para" being spread over three different tokens.

Additionally, we must take into account the nature of the freestyle genre. In many freestyle battles, one of the challenges consists in including a given word in the improvised verses. Said word may have nothing to do with the context of the battle. This could also potentially result in very low-frequency tokens appearing throughout the entire dataset, corresponding to these forcibly included words.

*2) Syllable level tokenization:* Syllable level tokenization allows overcome the issues of character-level tokenization to choose rhyming tokens, and it improved but not solved the problem of low-frequency tokens. With this approach, 4807 different tokens were obtained on the rap dataset, of which 1,170 appear only once (24.34% of the total) and 2,071 appear at most 3 times (43.06%). By using the freestyle transcripts dataset, 1911 different tokens are obtained with 479 of these appearing only once (25.07%) and 882 appearing at most 3 times (46.15%). In both datasets, we observe an improvement in the percentages of tokens with low frequency

over tokenization at the word level, but the values are still significantly high. Table III summarizes the observations made regarding this type of tokenization.

TABLE III
SUMMARY OF THE OBSERVATIONS MADE REGARDING THE AMOUNT OF TOKENS WITH LOW FREQUENCIES AT SYLLABLES LEVEL.

| Syllable Tokens | Rap DB | | Freestyle DB | |
|---|---|---|---|---|
| Total tokens | 4647 | 100% | 1877 | 100% |
| Tokens that appear once | 1208 | 26,00% | 465 | 24,77% |
| Tokens that appear at most 3 times | 2086 | 44,89% | 858 | 45,71% |

*3) Tokenization using Byte Pair Encoding:* The Byte Pair Encoding algorithm tokenizes a text identifying the tokens that appear most frequently, and therefore it can be useful to solve the low-frequency tokens problem. We ran the algorithm over different vocabulary sizes: 20,000 (value close to the number of tokens obtained by tokenizing at the word level), 10,000, 5,000 (value close to the number of tokens obtained by tokenizing at the syllable level), 3,000, and 1,000. A significant improvement was achieved in terms of the number of tokens with low frequency mainly when using a smaller vocabulary size. This is to be expected since with a smaller vocabulary size the tokens tend to be smaller, closer to the character level. When using a larger vocabulary these become more complex and usually correspond to whole words. Figure 3 shows an example of different levels of tokenization over a phrase taken from the data set.

*"En la improvisación*
*Lamentablemente yo muerdo como león"*

**Vocabulary size = 1000**:
*['en', 'la', 'improvis', 'a', 'ción', '[SEP]', 'la', 'm', 'enta', 'ble', 'mente', 'yo', 'muer', 'do', 'como', 'le', 'ón', '[SEP]']*

**Vocabulary size = 5000:**
*['en', 'la', 'improvisa', 'ción', '[SEP]', 'lamenta', 'ble', 'mente', 'yo', 'muer', 'do', 'como', 'le', 'ón', '[SEP]']*

**Vocabulary size = 20000:**
*['en', 'la', 'improvisación', '[SEP]', 'lamentablemente', 'yo', 'muer', 'do', 'como', 'león', '[SEP]']*

Fig. 3. Example of tokenization over a phrase from the data set using BPE with different vocabulary sizes

In order to evaluate this algorithm, we ran it on the entire dataset with each of the different sizes of vocabulary mentioned. This resulted in 5 different tokenizers, each with its set of tokens that correspond to respective vocabulary size. Then, with each of the tokenizers and sets of tokens defined, the number of tokens that appear only once and those that appear at most 3 times over the rap dataset and on the freestyle dataset, respectively, were evaluated. Results of this evaluation can be seen in figure 4 alongside the result for the other tokenization levels. As it was expected, the lowest percentages were obtained when using BPE with small vocabulary sizes, while as this size is increased the values get closer to those obtained when tokenizing at words or syllables level, especially when considering the database that contains exclusively transcripts of freestyle battles.


Fig. 4. Percentage of tokens that appear only once and tokens with at most 3 occurrences out of the total number of different tokens in the rap dataset (top) and freestyle dataset (bottom) when applying tokenization at different levels.

## III. Language Model and Training

In order to build the text generation system, we used a recurrent neural network architecture. The network receives as input a vector of integers that correspond to the indices of a sequence of tokens in the vocabulary; the length of the vocabulary is a hyperparameter. We carried out many experiments using different combinations of values for the vocabulary length and tokenizations.

The model has an embedding layer of dimension 300 (a standard value in other NLP works [16]) as the input layer and this is trained alongside the rest of the model. This is followed by a 256-unit bidirectional LSTM layer. We use a bidirectional layer since, as it we expand later in Section IV, text sequences used for training are given to the model in reverse order. Then, the LSTM units of the bidirectional layer that process the reverse sequence will be the ones that are capable of capturing the dependencies in the correct order of the text. Next, a Dropout layer was used to avoid overfitting, which in regards to text generation models results in generating only text similar to the existing one in the data set, instead of generalizing correctly for the generation of new texts. Finally,

a dense layer with the same size as the vocabulary followed by a softmax activation function is used as the output layer so the model predicts a probability distribution for the next word of the sentence given as input. Figure 5 illustrates the described model.

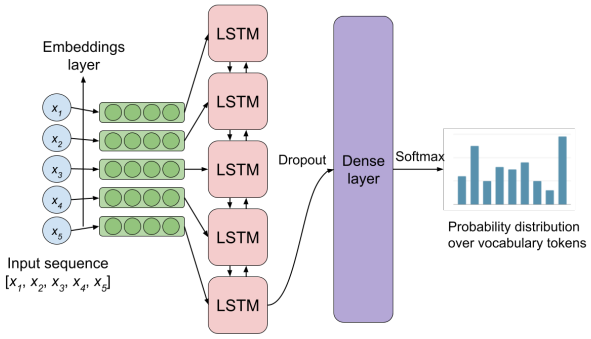


Fig. 5. Scheme of the language model used for text generation.

### A. Training

Using the described language model, different experiments were carried out varying on tokenization level and sequence lengths: at word level with sequences of 5 tokens, at syllables level with sequences of 8 tokens, and using BPE with a vocabulary size of 3000 and 5000 with sequences of size 6 and 5 respectively. BPE was not used with vocabulary sizes greater than 5000 since, as shown in Section II, when using these vocabulary sizes the presence of tokens with low frequency becomes a problem again. A vocabulary size of 1000 was not used either, since the tokens obtained using this size were too simple, very close to character level tokens.

Two token datasets were generated, one for each of the available databases. We excecuted a sliding window algorithm on the token databases to generate the training pairs $x, y$ where $x$ is a sequence of tokens that the model will use to try to predict the word $y$, represented as a one-hot vector. 80% of the pairs were used as a training set and 20% as a test set. Then, we pre-trained the model on the lyrics dataset and fine-tuned it on the freestyle battles' set. We used the Categorical Cross-Entropy loss function and data batches of size 256 for training the model.

Table IV shows a comparison of the results of the different training schemes. The pre-training on the database containing rap lyrics allowed to achieve high levels of accuracy over the freestyle dataset on many fewer training epochs. The pre-training consisted of 40 epochs when using BPE tokenization with vocabularies of size 3000 and 5000, in which they reached an accuracy of 85.98% and 82.58% over the training set, respectively. When training over a syllable level tokenization, more epochs were required to reach similar levels of accuracy: 80 epochs were carried out to reach a 78.14% of accuracy. Finally, the model trained using word-level tokenization obtained the highest accuracy on the training set: 95.35% after 50 epochs. After pre-training, between a quarter or a fifth of the number of training epochs on the set of freestyle transcripts were enough to achieve accuracy even greater than that achieved over the set of lyrics: The experiments using BPE with a vocabulary size of 3000 and 5000 reached an accuracy of 96.28 % and 95.89 % after 10 and 12 epochs respectively; the one that used syllables level tokenization reached an accuracy of 85.65% after 20 training periods and the one using word-level tokenization reached an accuracy of 97.71% after only 10 epochs.

TABLE IV
ACCURACY FOR DIFFERENT TOKENIZATION LEVELS IN BOTH DATASETS.

| Tokenization level | Word | Syllable | BPE(3000) | BPE(5000) |
|---|---|---|---|---|
| Sequence length | 5 | 8 | 6 | 5 |
| Training over song lyrics | | | | |
| Epochs | 50 | 80 | 40 | 40 |
| Accuracy over train set | 95.35% | 78.14% | 85.98% | 82.58% |
| Accuracy over test set | 42.42% | 60.89% | 57.28% | 52.06% |
| Training over freestyle transcriptions | | | | |
| Epochs | 10 | 20 | 12 | 10 |
| Accuracy over train set | 97.71% | 85.65% | 96.28% | 95.89% |
| Accuracy over test set | 40.31% | 59.81% | 55.51% | 51.12% |

The difference between the accuracy obtained in the training set and the test set for all the experiments is also striking. This difference is present in all the experiments, to a greater extent when training on the freestyle transcripts dataset. This behavior is typically associated with overfitting, however, it must be taken into account that the situation previously described about the presence of a large number of infrequent tokens in the dataset can also be a direct cause of this: several of the unique or low-frequency tokens are likely to be part of the sequences that make up the test set, which will result in the model not being able to learn about the use of those tokens from the training set. This could explain why word-level tokenization resulted in the highest difference between train and test set accuracy (95.35% and 42.42% respectively) while syllable-level tokenization got the best accuracy over test set (60.89%) followed closely by BPE using a vocabulary size of 3000 (57.28%). In any case, when using tokenization at a subword level, low-frequency words will generate sequences that will appear very few times, which may result in a phenomenon similar to what happened when using word-level tokenization. Either way, we believe that more training data is necessary to achieve an improvement in these results.

## IV. TEXT GENERATION

Once trained, a language model can be used to generate text from a seed simply by repeatedly using it as input for the network, choosing randomly a word from the generated probability distribution, and updating the seed with the chosen word. It is also a common practice to divide said probability distribution by a factor known as temperature which allows to soften the probability distribution and control the randomness

of the chosen word [17]. Although the generated text effectively replicates the desired style, the model could not learn some aspects correctly, such as the rhyme and the number of verses in each stanza. This can be observed in Figure 6, where we show a sample text generated using this method.

*Eso es verdad, es es que esto está difícil*
*Pero ey, yo te pateo cuando vengo te noqueo*
*Porque soy un teólogo, un psicólogo y porque vengo con empeño*

*Aparte, yo sé que tengo nivel*
*Mientras este se cree pablo neruda*
*Así que viene con su pluma*
*Pero yo hablo con la mente desnuda*

*Se quien soy, ven como juego?*
*Ves lo que hizo este ingenio, tené cuidado*

Fig. 6. Text generated using BPE tokenization with vocabulary size 5000, sequence length 5 and temperature of 1.5. The text was modified to capitalize the first letter of each verse, since the model was trained only with lowercase characters.

Therefore, we modified the generation method to guarantee that the stanzas contain the appropriate amount of verses. This resulted to be trivial because of the token "[SEP]" that identified the end of each verse. The token allowed to identify when the model finished generating a stanza by simply counting the number of "[SEP]" tokens, which was fixed to 4 before starting a new one. Figure 7 shows some examples of text generated after fixing the verse amount for each stanza.

*En cada rima, soy de los mejores*
*Y le doy gracias a las batallas*
*Por hacerme ser alguien inteligente*
*Que sabe pararse de frente*

*Sabes que pasa? Que no tenés fluidez*
*Te lo aclaro, mi hermano*
*Nadie confía en vos*
*Nadie quiere saber de tu improvisación*

*Y bueno de verdad, lo mato en la improvisación*
*Te falta aprender que tengo creación*
*Porque no tenés el nivel*
*Te subes a mi montaña rusa de este parque de diversión*

Fig. 7. Text generated after fixing the verse amount for each stanza and using the same hyperparameters as in figure 6.

### A. Rhyming

Ensuring that each of the verses adapts to a corresponding rhyme structure is not as trivial as fixing the number of verses. The difficulties that have to be dealt with to achieve a correct rhyme structure are:

1) Identifying when two words rhyme, by contemplating several possible cases of accentuation and taking into account particular rules of the language to identify their respective stressed vowels. This becomes even more complex with the appearance of words in English, expressions, and onomatopoeias that do not follow the corresponding Spanish accentuation rules. For example, the expression "yeah", taken from English, should either be written in Spanish as "yea", or have an accent mark in it. In this situations, you can't rely on accentuation rules.
2) For the text to fit a particular rhyming structure it is necessary to focus on the generation of the last word of each verse. However, as the number of words/syllables per verse is not fixed and the next word is chosen randomly from the probability distribution, a forward generation scheme cannot condition the model on the last word until it has already been generated.
3) Modifying the last word of a verse after having generated the previous ones to adapt it to a rhyme structure can lead to incoherent phrases. For example, if we had to change the ending word of the phrase "I bite like a lion" by only considering that it must rhyme with another word, it would likely result in an incoherent verse, as the context strongly implies that the last word must be, in this case, an animal.
4) Finally, when using word-level tokenization, computing whether two words rhyme is relatively easy. Therefore, the output probabilities of the model can be modified to increase the probability of rhyming words and decrease the rest. This approach, however, becomes much more complex when using subword level tokenization. When working at syllable level or BPE, it is not enough to look at a particular token to know if it will be part of a word that rhymes with the desired one. Therefore, the predictions of one or more future tokens must be analyzed, which is computationally expensive.

In the following subsections, we describe our proposed solution to these problems.

*1) Obtaining rhyming pairs:* To generate text under a certain rhyme structure, we must identify in a pair of words the presence or absence of rhyme, assonance, or consonant. Most similar works for the English language use rhyming dictionaries for this [8] [9]. However, there is no similar construct for Spanish at the time this work was carried out. For this reason, we developed an algorithm to recognize consonant or assonance rhymes between two words according to Spanish accentuation rules.

However, as already mentioned, several of the expressions in the text are either English words or expressions, so the algorithm wouldn't work with these words. To solve this, we also defined a mapping from each of the most frequently found English expressions in the songs to its corresponding Spanish phonetic spelling. The phonetic spelling is an adaptation of the English word to the idiomatic rules of Spanish (for example, for the word "flow" is assigned "flou" and for "freestyle", "fristail"). In this way, we can identify when English words rhyme between them or with another word in Spanish. Figure 8 shows a subset of the dictionary which is available in the project's repository.

*2) Rhyme forcing with a rhyme model:* As described before, by having a way to identify when two words rhyme, we can modify the text generation probabilities for the last word of

```
``flow": ``flóu",
``freestyle": ``fristail",
``disney": ``dísney",
``baby": ``beibi",
``destroy": ``distróy",
``yeah": ``yea",
``hardcore": ``hárdcor",
``street": ``estrit",
``beat": ``bit",
``hater": ``jeiter"
```

Fig. 8. Samples from the phonetic English-Spanish dictionary.

the verse to force it to rhyme. In this case, we increased the probabilities by different factors, depending on if it is a consonant or assonance rhyme, and the word is sampled from this modified distribution, so it is highly likely that it would rhyme with the desired verse.

Figure 9 shows three stanzas generated as described above in which words chosen from the modified probability distribution are in bold. For each word in bold, it also shows the word that was supposed to be chosen if the distribution wasn't modified below each stanza. Although the verses now effectively rhyme, the words taken from modified probability distributions are generally out of context. This happens in practically all these verses: in the last verse of the first stanza shown, the phrase "you are not going to win" is much more coherent than "you are not going to shave" and even more in the context of a freestyle battle. The same applies in the last verse of the second stanza, where "I teach the class, I sit you at the desk" is much more coherent than the final sentence: "I teach the class, I sit you at the mall" and many more of the verses.

*Que pueda hacerlo en un momento como este ya **apremia***
*Que vas a hacer? Es normal que yo lo **bacterias***
*Claro que esta batalla ya la **bacteria***
*No tenés lo mismo, no vas a **afeitar***

*("adorne"→"**bacterias**"), ("regaló"→"**bacteria**"), ("**ganar**"→"afeitar")*

*Te doy, te gano, aquí te voy a **dar***
*Con las rimas que suelto **ja'***
*Por escribir esos **ah***
*Yo doy la clase, te siento en el **mall***

*("épicas"→"**ja'**"), ("rumores"→"**ah**"), ("pupitre"→"**mall**")*

*Se cubren los oídos, como por **dinamita***
*Pero yo voy a secuestrar completa a **imaginas***
*Resentido! Y la tengo aunque ni siquiera lo **cristalina***
*No, no, sólo ven que no quedóimaginar*

*("Afrodita"→"**imaginas**"), ("veas"→"**cristalina**"), ("ninguno"→"**imaginar**")*

Fig. 9. Stanzas generated as described using word level tokenization. Below each word, we show the word originally predicted for each of the verse endings, followed by the word chosen after modifying the probability distribution to force words to rhyme.

*3) Inverted text generation:* To solve the problem presented before, we apply a reverse text generation scheme, Instead of training the model to predict the next word of a sequence, we train it to predict the previous one. This approach was used in works with similar domains such as [10] and [9] allows to know exactly when the next word to predict is the last word of a verse (after the appearance of a token "[SEP]") and to modify the word to chose without generating incoherences within the verse in which it is found since it will be starting from the chosen word.

Similarly to figure 9, Figure 10 shows the application of the reverse generation algorithm. The words that had to be adjusted so that the stanza complies with the rhyme structure no longer represent coherence problems for the verses and are much more coherent than those that were to be chosen before modifying the probability distribution.

This is only possible because the verse was generated after replacing the word. However, the words that were to be chosen in many cases were more closely related to the next verse in the stanza (which was generated immediately before the word). For example, in the third verse of the first stanza the word "roman", which was directly related to the theme of the following verse, was replaced by the word "demand". Although "demand" fits correctly the verse in which it is found, is not related to the next one, which was generated before the word. This can also be observed in the second verse of the second stanza when replacing "dead" with "territory".

*Y seguro que se acabaron las **entradas***
*Porque mi faceta te hace ser un cometa, no cometas más **erratas***
*Yo represento a la gente que demanda*
*Este es el imperio romano, bienvenido a **esparta***

*("peligro"→"**entradas**"), ("chaval"→"**erratas**"), ("romano"→"**demanda**")*

*Vengo con rap, rap en **episodio***
*Represento a este chile hermano, **territorio***
*Y cuando yo ya arranco tu te **mueres***
*La verdad que tengo el don y un hijo de **nueve***

*("latina"→"**episodio**"), ("muerto"→"**territorio**"), ("encontrar"→"**mueres**")*
*Ha vuelto este ritmo que parece grato*
*Hoy hoy día para mí todos esos rivales son malos*
*Saben que lo hago yo lo hago cuando hablo*
*Oh, de verdad, aprovecha de aprenderte el año*
*("lindo"→"**grato**"), ("lenguaje"→"**malos**"), ("lejos"→"**hablo**")*

Fig. 10. Stanzas generated as described using word level tokenization and reverse generation.

*4) Subword level rhyming:* As mentioned before, is not possible to apply the described algorithm when using subword level tokenization as it is impossible to know from a single subword token if the word it will be part of rhymes with another given word, even if it is the last token of that word. So, to make this scheme work with subword level tokenization, a rhyming dictionary was generated by computing all the rhyming words for each word in the vocabulary and if the rhyme is consonant or assonance. The generation of the rhyming dictionary is computationally very expensive since it is of quadratic order over the vocabulary size (28,424).

However, this is computed only once and does not slow down the generation of the text later on.

Therefore, the modified version of the algorithm to generate a verse that rhymes with a given word $w$ using subword level tokenization and reverse generation works as follows:

- First, the rhyming words for $w$ are obtained through the rhyming dictionary, and for each of these words $p$:
  - $p$ is tokenized resulting in the sequence $[t_1, t_2, ..., t_n]$.
  - The probability of generating $t_n$ ($P(gen(t_n))$) is obtained from the probability distribution produced by the model when processing the previous sequence $s$. Then, the probability of generating each of the tokens $t_i$ having generated the token $t_{i+1}$ ($P(gen(t_i)|gen(t_{i+1}))$) is obtained adding the token $t_{i+1}$ to the end of $s$ and using this as input for the model. ($P(gen(t_i)|gen(t_{i+1}))$) then corresponds to the probability for the token $t_i$ assigned by the model.
  - Then the possibility of generating word $p$ is:

$$P(gen(p)) = P(gen(t_1)|gen(t_2) \cap ... \cap gen(t_n)) \tag{1}$$

And because of the chain rule for probability:

$$P(gen(p)) = \prod_{k=1}^{n} P(gen(t_k)| \bigcap_{j=n}^{k-1} gen(t_j)) \tag{2}$$

  - $P(gen(p))$ is computed and stored in a dictionary using $p$ as key.
- After this, we end up with a dictionary containing, for each word that rhymes with the given word $w$, its probability of being generated next. We then sample a word from that dictionary instead of the entire vocabulary, guaranteeing that it rhymes.

### B. Generation algorithm

Finally, the process of generating a stanza that conforms to a certain rhyme scheme consists of:

- Entering a text that is used as a seed that is tokenized using the corresponding tokenization level. In case that it contains tokens that are not in the vocabulary, those are assigned an unknown token ("[UNK]").
- Defining a rhyme structure randomly among some of the most typically used structures. In this work AAAA, AABB, ABAB, and ABBA structures were considered.
- Using the seed as input for the model and generating text until a token "[SEP]" is obtained indicating the end of the verse.
- The next word to generate will be the last word of the previous verse, so it must be ensured that it follows the rhyming scheme chosen. In case that it has to rhyme with a word from a previous verse, the algorithm described in the previous subsection will be used to obtain a word that meets the expected rhyme.
- This will be repeated until the four lines are completed according to the rhyme scheme of the stanza.

Figure 11 shows some verses generated using the algorithm described.

*De la esquina cómo verás por más que me ignoraban*
*¿Y dónde está todo eso con lo que soñabas*
*Hago la diferencia con mi poesía pero nada*
*Yo, yeah, entendí que todo importaba*

*La diferencia con el es que me gano el primer puesto*
*La fé mueve montañas mis montañas*
*Mi personalidad que me acompañaba*
*Mirame a mi ya me voy yendo*

*Yo sí te voy a subir el pulgar*
*Es caro, flow extranjero*
*Representado canotes a llorar*
*Vas a perderlo, negro, no retrocedemos*

*Mientras yo escribía vos la criticaste*
*Que si matan, que si roncan, si te tocan los parto como street fighter*
*Quiero verte bailar con los diamantes*
*No de versace na' nadie lo hace*

Fig. 11. Samples of stanzas generated using the described algorithm and a model trained using BPE with a vocabulary size of 5000, sequence length 5 and temperature 1. Both first and last stanzas were generated under the AAAA rhyme scheme, while the second and third follow the ABBA and ABAB schemes respectively.

## V. Conclusions and Future Work

In this work, we propose a text generator system that generates lyrics that are framed within the genre of freestyle battles. By using a neural language model and text generation algorithms, the system respects the corresponding structure and rhyme.

In order to train the model, it was necessary to create a database that specifically contains transcripts of freestyle battles, since no other public database containing this type of material was found at the time of writing this paper. In addition, another larger database that contains lyrics of rap songs was also generated and used to pre-train the language model.

We compared different tokenization approaches to evaluate their applicability to train a language model with scarce data and a very diverse vocabulary. Tokenization at the word level may not be ideal as it resulted in a large number of tokens appearing only once in the text or having very low frequency. Tokenization at the subword level proved to be more suitable; at the syllable level, it did indeed reduce the number of tokens with low frequency, but still represented a significant portion of the total. The Byte Pair Encoding algorithm with small vocabulary sizes (1000, 3000, or to a lesser extent 5000) was the most efficient tokenization method to face this problem, resulting in the number of tokens with less than three occurrences in the whole set of data being around only 1% of the total tokens.

The use of pre-training fine-tuning on the two databases proved effective for training the model. After a pre-training on

the larger database, an accuracy of about 90% was achieved over the freestyle battles' dataset with significantly fewer training epochs.

Finally, we observed that the language model was able to effectively capture the style of the genre, but not the correct structuring or use of rhyme. However, using a modified model that guaranteed that the generated text meets these features, it was possible to generate text that fits not only the style but also the structure of freestyle battles.

Possible future lines of work to follow include developing a system that, following what happens in freestyle battles, is capable of not only generating text but also responding to a verse produced by a potential opponent. We also considered important to increase the amount of data available for training the models, either simply by expanding the size of the datasets or using data-augmentation techniques. Including the use of few-shot learning techniques as fine-tuning could help to make the most of the currently available datasets, as well as identifying tokens that refer to the same word (as what happened with "para", "pa"' y "pa") and taking that into account while training. Finally, it would be interesting to experiment with other neural network architectures like Transformers to build the language model and compare the results with the current approach.

## REFERENCES


[1] L. Deng and Y. Liu, *Deep learning in natural language processing*. Springer, 2018.

[2] D. Jurafsky and J. H. Martin, "Speech and language processing (draft)," *Chapter A: Hidden Markov Models (Draft of September 11, 2018). Retrieved March*, vol. 19, p. 2019, 2018.

[3] Y. Goldberg and G. Hirst, "Neural network methods in natural language processing. morgan & claypool publishers(2017)," *9781627052986 (zitiert auf Seite 69)*, 2017.

[4] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, 1st ed. O'Reilly Media, Inc., 2009.

[5] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[6] J. Mackenzie, R. Benham, M. Petri, J. R. Trippas, J. S. Culpepper, and A. Moffat, "CC-news-en: A large english news corpus," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, ser. CIKM '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 3077–3084. [Online]. Available: https://doi.org/10.1145/3340531.3412762

[7] G. Branwen, "Gpt-3 creative fiction," 2020.

[8] M. Ghazvininejad, X. Shi, Y. Choi, and K. Knight, "Generating topical poetry," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 1183–1191.

[9] J. H. Lau, T. Cohn, T. Baldwin, J. Brooke, and A. Hammond, "Deepspeare: A joint neural model of poetic language, meter and rhyme," *arXiv preprint arXiv:1807.03491*, 2018.

[10] X. Yi, R. Li, and M. Sun, "Generating chinese classical poems with rnn encoder-decoder," in *Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data*. Springer, 2017, pp. 211–223.

[11] A. Zugarini, S. Melacci, and M. Maggini, "Neural poetry: Learning to generate poems using syllables," in *International Conference on Artificial Neural Networks*. Springer, 2019, pp. 313–325.

[12] P. Potash, A. Romanov, and A. Rumshisky, "Ghostwriter: Using an lstm for automatic rap lyric generation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1919–1924.

[13] P. Edwards, *How to rap*. Chicago Review Press, 2009.

[14] Batallas de rap lyrics wiki. [Online]. Available: https://batallas-de-rap-lyrics.fandom.com/es/wiki/Batallas_de_Rap_Lyrics_Wiki

[15] MIDUSI. Freestyle lyrics generator. [Online]. Available: https://github.com/midusi/freestyle_generator

[16] Z. Yin and Y. Shen, "On the dimensionality of word embedding," in *Advances in Neural Information Processing Systems*, 2018, pp. 887–898.

[17] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.