GPoeT: a Language Model Trained for Rhyme Generation on Synthetic Data

Andrei Popescu-Belis[1,2], **Àlex R. Atrio**[1,2], **Bastien Bernath**[2],
Étienne Boisson[2], **Teo Ferrari**[1], **Xavier Theimer-Lienhard**[2] **and Giorgos Vernikos**[1,2]

[1]HEIG-VD / HES-SO, Yverdon-les-Bains, Switzerland
[2]EPFL, Lausanne, Switzerland
{andrei.popescu-belis, alejandro.ramirezatrio, georgios.vernikos}@heig-vd.ch

Abstract

Poem generation with language models requires the modeling of rhyming patterns. We propose a novel solution for learning to rhyme, based on synthetic data generated with a rule-based rhyming algorithm. The algorithm and an evaluation metric use a phonetic dictionary and the definitions of perfect and assonant rhymes. We fine-tune a GPT-2 English model with 124M parameters on 142 MB of natural poems and find that this model generates consecutive rhymes infrequently (11%). We then fine-tune the model on 6 MB of synthetic quatrains with consecutive rhymes (AABB) and obtain nearly 60% of rhyming lines in samples generated by the model. Alternating rhymes (ABAB) are more difficult to model because of longer-range dependencies, but they are still learnable from synthetic data, reaching 45% of rhyming lines in generated samples.

1 Introduction

The quality of texts generated by language models (LM) has improved tremendously in recent years. While their factual accuracy is still open to debate, this is not an issue when using LMs with a creative purpose, in particular to generate works of art such as poems. In the recent past, LMs were put to use for poetry generation in several studies (Hopkins and Kiela, 2017; Lau et al., 2018; Van de Cruys, 2020; Wöckener et al., 2021; Uthus et al., 2022; Ormazabal et al., 2022), which found that fluency and intelligibility reached satisfactory levels. However, poems often exhibit structural, text-level properties that are still quite difficult to manage by LMs: rhyming patterns and division into verses and stanzas. While not all poems make use of these properties, a convincing LM for poetry generation should be able to deal with them.

In this paper, we focus on the first property and propose a method to adapt an LM so that it generates rhyming verses, with modest computing requirements. We start from an unconstrained autoregressive LM, in our case GPT-2, which we fine-tune first on a poetry corpus of about 120 MB to improve its style (Section 3). We design a rule-based system which modifies text generated by the LM so that it obeys a given rhyming pattern while retaining acceptable fluency, and we generate two datasets of 160k lines (6 MB) each with the AABB and ABAB patterns (Section 4). We further fine-tune the LM on these synthetic datasets in order to generate rhyming verses with the respective patterns, thus showing that they can be learned by a moderately-sized LM (Section 5).

We also introduce a rhyming metric (see Section 2) based on an English rhyming dictionary, and use it throughout the study to count the proportion of perfect and assonant rhymes generated by a model. We find that this is very low (11%) for the LM fine-tuned on natural poetry with variable rhyming patterns, but increases to around 60% when the LM learns only the AABB pattern from synthetic data. The ABAB pattern is more challenging, but can still be learned, reaching around 45% rhyming lines. In the conclusion (Section 7), we discuss some issues related to the integration of the rhyming LMs into an existing, operational system for interactive poetry generation.[1]

Our contributions are the following:

- a metric computing how many lines have perfect or assonant rhymes that conform to a given pattern in English;
- a rule-based algorithm to generate rhyming lines of a given pattern, based on a GPT-2 LM fine-tuned on poetry;
- a demonstration that even medium-scale LMs can be fine-tuned to learn a rhyming pattern from machine-generated poems;
- evidence that local rhyming patterns are more easily learned than those implying longer-range dependencies.

[1] Source code available at github.com/heig-iict-ida/crpo.

Proceedings of the 7th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature (LaTeCH-CLfL2023), pages 10–20
May 5, 2023 ©2023 Association for Computational Linguistics

2 Measuring the Number of Rhymes

A criterion for measuring the number of rhyming verses is key for the present study. We present a metric that distinguishes between perfect rhymes, assonant rhymes, and no rhymes, using a rhyming dictionary derived from an English pronunciation dictionary. We test it on a corpus of human poetry annotated for rhyme and show that its accuracy is sufficient for use in this study.

2.1 Definitions of Rhymes

Following a widespread definition,[2] also adopted by Van de Cruys (2020), a perfect rhyme is the identity of the final vowel and consonant sounds of a word, starting with the first vowel of the last stressed syllable. An assonant rhyme is the identity of the final vowels in the last stressed syllable, but not of the ending consonant.

Since the addition of stress information would reduce the amount of available candidates for a rhyme, we simplify the definition of a rhyme between words w_1 and w_2 as follows, using the phonetic representation of each word $phon(w)$.

1. We have a *perfect rhyme* if $phon(w_1)$ and $phon(w_2)$ end with the same vowel followed by the same consonant(s), if any.
2. We have an *assonant rhyme* if $phon(w_1)$ and $phon(w_2)$ end with the same vowel, followed by one or more non-identical consonants.
3. Otherwise, the lines *do not rhyme*.

2.2 Construction of the Rhyming Dictionary

To apply the preceding definitions, and to generate rhymes according to them, we build a rhyming dictionary starting from the Carnegie Mellon Pronouncing Dictionary of English.[3] The dictionary contains pronunciations of 123,631 English words. Each word is associated with a series of phonemes coded using ASCII letters only, for example '`K AE M P EY N`' for the word 'campaign'.

We distinguish 15 phonemic vowels (e.g., 'AH', 'AW', 'EY', 'OY') and consider all other phonemes as consonants. To each word from the dictionary we associate two strings.

1. The last phonemic vowel and all the consonants following it (if any), to allow testing for perfect rhymes.
2. The last phonemic vowel only, whether it is followed or not by consonants, to allow testing for assonant rhymes.

Examples of entries in our rhyming dictionary are therefore ('campaign' → 'eyn', 'ey'), ('copycodes' → 'owdz', 'ow'), ('vanilla' → 'ah', 'ah'), ('do' → 'uw', 'uw'), and ('wouldn't' → 'ahnt', 'ah').

To help with rule-based generation of rhymes, we create two dictionaries that invert the first one, for efficiency reasons. One has the strings defining the perfect rhymes as keys and the corresponding words as values – for instance ('eyn' → . . ., 'campaign', 'overtrain', 'plane', . . .) – and the other one has the strings defining the assonant rhymes as keys and the corresponding words as values. The first additional dictionary has 1,356 keys (word endings for perfect rhymes) and an average number of 91 words per key, while the second one has only 15 keys (the number of phonemic vowels) and an average of 6,507 words per key, ranging from 576 to 34,037.

2.3 Definition of the Metric

The proposed metric for rhymes follows from the definitions above, and makes use of the first dictionary. Given two words – the ending words of two lines of poetry – we compare their entries in the dictionary. If the first strings are identical, then we count a *perfect rhyme*. If they are not, we examine the second strings, and if they are identical, then we count an *assonant rhyme*. If not, then we consider that the words do not rhyme. The order of testing is important, because for words ending with a vowel, such as ('vanilla' → 'ah', 'ah') and ('Godzilla' → 'ah', 'ah'), both entries match, but we want to consider this as a perfect rhyme.

To apply the metric, the lines of the poem are first tokenized using NLTK's `word_tokenize()` function.[4] If a line finishes with punctuation, we discard it and examine the last word of the line. If the line ends with a contraction (such as 'wouldn't') we join back the two resulting tokens generated by `word_tokenize()`. If a word does not appear in the pronunciation dictionary, then we search for the most similar one in terms of string edit distance using the `get_close_matches()` function from the 'difflib' Python package (a time-consuming operation). We experimented with restricting the similarity search to the initial parts of words, because changing the end changes the rhyme, but did not

[2] See e.g. rhymenow.com/types-of-rhymes.

[3] Freely available from svn.code.sf.net/p/cmusphinx/code/trunk/cmudict/sphinxdict/cmudict_SPHINX_40.

[4] From www.nltk.org.

		Our metric		
		Perfect rhyme	Assonant rhyme	No rhyme
Human annotation	Rhyming	27,174 (78.8%)	680 (2.0%)	6,628 (19.2%)
	Not rhyming	4,209 (1.3%)	25,163 (7.9%)	290,633 (90.8%)

Table 1: Confusion matrix for rhyming detection by our metric vs. human annotation.

observe significant differences when validating the metric.

2.4 Validating the Metric

We validated our metric on the Chicago Rhyming Poetry Corpus[5] which includes English poems annotated with their rhymes. For each poem, the annotation marks the last word of each line with an index number, and co-indexes rhyming words. For instance, a three-line stanza could be annotated as “house pain souse” followed by “1 2 1”, indicating that its lines end respectively with the words ‘house’, ‘pain’ and ‘souse’ and that the first line rhymes with the third one.

From the corpus, we derive ground-truth pairs of rhyming and non-rhyming words. For each annotated stanza with k line-ending words, we consider all $k(k-1)/2$ pairs of words and separate them using the annotations in rhyming or non-rhyming pairs. During this process, we found a small number of annotation inconsistencies, and we checked how many words are actually present in our pronunciation dictionary. As for some poets the total number of unknown words is quite high, we exclude them from the dataset, on the grounds that their vocabulary or spelling is too different from modern use.[6]

In fact, the human assessment of rhymes may not be 100% reliable, due to the evolution of pronunciation and the imperfections of the annotation process. Additionally, some pairs annotated as non-rhyming may in fact rhyme, but have not been annotated as such since they do not fit the rhyming schema of the poem. The creation of a validation corpus can thus be improved, but the goal is to obtain the most reliable rather than the largest possible dataset, in order to validate the metric. Overall, we obtained 34,482 rhyming word pairs and 320,005 non-rhyming ones.

We assessed if our metric, given each word pair, can correctly label it as rhyming or non-rhyming. As the metric distinguishes perfect from assonant rhymes, we may or not merge these two categories. Results are shown in Table 1. If we merge perfect and assonant rhymes, our metric finds 80.8% of the rhymes (most of them perfect) but also labels 9.2% of non-rhyming words as rhyming ($F_1 = 0.61$). To maximize the F_1-score, it would seem preferable not to count assonant rhymes (then $F_1 = 0.83$) but in what follows we will count both types of rhymes.

Upon inspection, recall errors are often due to words that are absent from the pronunciation dictionary, and when replaced with similarly-spelled ones, their pronunciations differ. For instance, ‘marinere’ → ‘mariner’ no longer rhymes with ‘hear’, or ‘thro” → ‘throw’ no longer rhymes with ‘flew’. In other cases, the pronunciation in our dictionary does not match the one considered by the poet: ‘stood’ rhymes with ‘blood’ and ‘thus’ rhymes with ‘albatross’ according to the corpus, but not in our dictionary. As for precision errors, a large part of them are assonant rhymes which are not annotated in the corpus. For instance, ‘there’-‘around’-‘howl’d’-‘swound’ is annotated as ABCB but we detect an assonance because the last three words have the same final vowel. Finally, annotation mistakes in the corpus can lead to both types of errors, e.g. ‘close’-‘beat’-‘sky’-‘eye’-‘feet’ is annotated as ABCCC in the corpus but correctly labeled by us as ABCCB.

3 An Auto-regressive Language Model Fine-Tuned on Poetry

Our starting point is GPT-2 (Radford et al., 2019), a general-purpose decoder LM for English. We use the Python implementation provided by the Huggingface library (Wolf et al., 2019).[7] We enable the model to generate poetry by fine-tuning it first on a corpus of English poetry (3.1), and then by designing constraints so that its output has the form of a poem, with lines and stanzas (3.2). We evaluate the frequency of rhymes in the output of this model using our metric (3.3), before moving

[5] github.com/sravanareddy/rhymedata

[6] These are, by decreasing numbers of unknown words: Spenser, Lovelace, Drayton, Jonson, Kipling, and Byron.

[7] huggingface.co/gpt2

on to its specific training for rhyming in the next sections.

3.1 Fine-tuning GPT-2 on Poetry

We use the *Gutenberg Poetry Corpus*[8] composed of approximately 3 million lines of poetry extracted from hundreds of poetry books from Project Gutenberg. Unlike the Chicago Rhyming Poetry Corpus used for validation in Section 2.4, we do not filter out any author. We convert the corpus from the JSON format it into raw text, with poetry lines separated by newline characters ('\n') and no blank lines. Therefore, all information about stanzas, poems and books is removed, and we also delete quotation marks and dashes. However, to emphasize the importance of lines, we prefix each line with a '<start>' tag, which will help generation. The result is a text file with 3,085,063 lines (142 MB). On this data, we fine-tune the smallest GPT-2 model (124M parameters) for three epochs, which takes ca. 3 hours on a single Nvidia GeForce RTX 3080 GPU.

3.2 Setting the Poem's Form

Generating text in a form that is typical of poetry is essential for considering rhyming patterns because without a division into lines (verses) there are no line endings that can rhyme. A general discussion of form constraints is out of the scope of this paper (see Section 4.1 of Popescu-Belis et al., 2022), and we summarize the approach as follows.

We give the desired structure of the poem – number of stanzas, number of lines in each stanza, and number of syllables in each line – to the following algorithm. The first two parameters are easy to constrain, by inserting one or two newline characters. However, it is harder to constrain GPT-2 to generate a pre-specified number of syllables in a line. We generate the poem line by line, with decoding by sampling according to the word probability generated by GPT-2, modulated by a temperature factor. To generate line k, we provide GPT-2 with lines $1, 2, \ldots, k-1$ as context. To obtain the expected number of syllables S_E in line k, we loop through the following steps:

1. Require GPT-2 to generate a line L with a fixed number of tokens, computed from S_E using a ratio of 1.5 syllables per token.[9]
2. Count the actual number of syllables S_L of the line L, using an algorithm for English by Emre Aydin (found at eayd.in/?p=232).
3. Exit the loop with L if $S_L = S_E$, or after 10 iterations.

3.3 Number of Rhymes of the Baseline

Using the GPT-2 model fine-tuned on poetry, we evaluate the number of rhyming verses as a term of comparison with further models. As we cannot make any prior assumption on the rhyming pattern, we simply group the generated verses into pairs (or couplets) by inserting a newline every other verse. When applying our metric to a set of 4,000 couplets generated in this way, we find that only 4.3% have perfect rhymes, while 6.6% have assonant rhymes, and the remaining 89.1% do not rhyme at all.

4 Synthetic Data with Rhymes: Rule-based Generation

We use a rule-based approach to modify the poems generated by the previous model so that they follow a given rhyme scheme, which is specified in conventional form (e.g. AABB, ABAB or ABBA). This is part of our earlier interactive system for poetry generation (Popescu-Belis et al., 2022) which combines LMs with rules governing form, rhymes, topics and emotions.

The rule-based rhyming algorithm parses the scheme, and for every second line of a rhyme (e.g., given AABB, for the second and fourth lines), it modifies the last word so that it rhymes with the last word of the previous line. The inverted rhyming dictionaries presented in Section 2.2 and the fine-tuned GPT-2 model are used as follows.

The algorithm obtains from the first dictionary the perfect rhyme ending the word to replace, and it searches the second dictionary for all the words that share this perfect rhyme. If none is found, the words sharing the respective assonant rhyme are used instead. Each word is inserted in the entire line and the result is submitted to GPT-2, which generates a likelihood score for each of these sequences. The replacement word leading to the highest score is selected. Therefore, to generate rhyming poems, we first generate a non-rhyming one and then we re-generate the last words so that they rhyme according to the given patters.

Using this strategy, we generate large numbers of poems, first with the AABB rhyming pattern, and later with the more challenging ABAB pattern. For

[8] github.com/aparrish/gutenberg-poetry-corpus

[9] Technically, the decoder is given a *maximum* length, but in practice, we never observed end-of-sequence symbols, so this length is always reached.

each pattern we generate 20,000 quatrains (four-line stanzas) resulting in about 6 MB of text. Some cleaning of the data is necessary because some lines are made mostly of punctuation or include special characters. About 0.04% of the lines are removed. To simplify training, we insert a blank line after lines AA and then BB of the quatrain, so that the training data is made of rhyming couplets only. Alternatively, to learn ABAB, we insert a blank line after each quatrain. Our metric found that the first dataset has a rhyming accuracy of 97.8%, which is expected because the rhyming algorithm and the metric make use of the same dictionary.

Moreover, as the LM must capture dependencies between words at the end of lines regardless of the punctuation, we hypothesize that if we remove punctuation at the end of the verses in the training dataset, the LM would better learn rhyming patterns. The results below confirm this hypothesis.

5 Learning Rhyming Patterns from Synthetic Data

5.1 Learning the AABB Pattern

Our first experiment with fine-tuning GPT-2 on synthetic data studies the simplest rhyming pattern, where two consecutive lines rhyme. As stated above, the synthetic data is made of couplets, and this is what we expect the fine-tuned model, called GPoeT, to generate as well.

To measure the proportion of rhyming verses, we consider only the couplets and exclude isolated lines, or stanzas with an odd number of lines. This ensures that we always test the rhyming of paired lines in the sample data. During fine-tuning, we generate ca. 50 kB of text every 10 epochs and measure the proportion of rhyming lines on this sample.[10] Cleaning the isolated lines removes ca. 20% of the text, a number which stays quite constant during fine-tuning (red curve in Figures 1 and 4). In other words, the model produces couplets in 80% of the cases.

The evolution of the rhyming capabilities of GPoeT during fine-tuning is shown in Figure 1. The improvement with respect to the baseline (fine-tuned on the Gutenberg Poetry Corpus only) is very substantial, from a proportion of perfectly rhyming couplets of 4.3% to 56.2% (a factor of 13). When counting both types of rhymes, GPoeT generates 59% of rhyming couplets vs. 7.6% for the baseline (a factor of 7.7). The proportion of perfect rhymes rises quickly and then converges to around 56% after 70 epochs, while the proportion of assonant rhymes remains quite constant, likely because the data used for fine-tuning has only perfect rhymes. From the evolution of the curves, the system has likely reached its maximal performance.

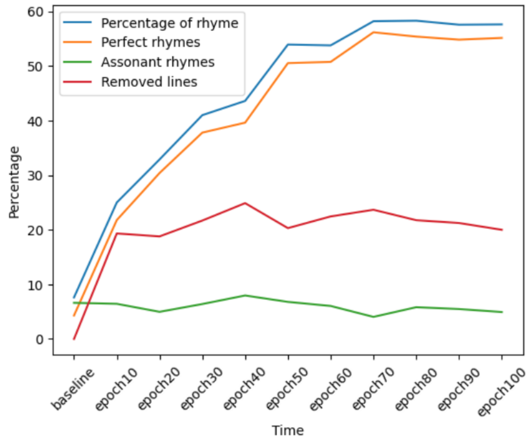

Figure 1: Proportion of perfect and assonant rhymes generated during the fine-tuning of GPoeT on AABB synthetic data, for 100 epochs.

The learning rate decreases linearly with the number of steps, from 5×10^{-5} to 9×10^{-7} along 10 epochs. After 10 epochs we reset the learning rate to the initial value. In this way, we force larger updates of the parameters at regular time intervals, which makes the model more robust, following our insights from low-resource machine translation (Atrio and Popescu-Belis, 2022). This may improve training, as opposed to a learning rate that decreases too quickly. We can see in Figure 2 that the validation loss globally decreases over time, with small increases every 10 epochs when the learning rate is reset.

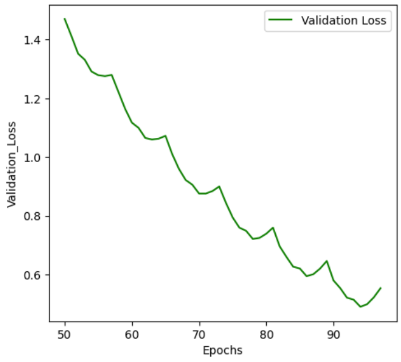

Figure 2: Evolution of the validation loss while learning the AABB pattern.

[10] On one GPU, 10 epochs take about 25 minutes.

We validate the use of quatrains stripped of the final punctuation for training, hypothesizing that such tokens may hinder the learning of rhymes. We compare the proportion of rhymes generated by GPoeT after fine-tuning for 10 epochs on the synthetic quatrains when the final punctuation is kept *versus* deleted. The results shown in Table 2 confirm that deleting the punctuation from the training data is beneficial, and GPoeT was trained beyond 10 epochs on this data only.

	Final punctuation	
Metric	**kept**	**deleted**
Perfect rhymes	13.8%	18.4%
Assonant rhymes	8.1%	7.2%
No rhyme	78.1%	74.4%

Table 2: Scores after 10 epochs on fine-tuning on data with or without punctuation at the end of the lines.

We also experiment with a promising approach for accelerating fine-tuning. We alternate between (1) training on the full synthetic dataset for 20 epochs, and (2) training on a dataset containing only the last word of each line (i.e. pairs of rhyming words) for 10 epochs. The second stage is much quicker, and as the obtained scores are similar, we believe that training only on the rhyming words of lines should be studied in more detail in the future.

5.2 Sample Outputs of GPoeT

We provide below two unedited excerpts selected from the sample generated by the last GPoeT checkpoint.

The prince of men in arms he heard
So bold, so bold the warrior plundered

That she herself in sorrow cried
My God! who made the earth so bide

She sees no other sun above
Nor in that cloudless sky doth dove

My God! who made the earth so fair
And on this cloudless night hath mair

To the sound of your sweet voice
As of a little bird at choice

As in a trance the dreamer hears
At length a voice, so deep, so here's

That in itself it seems a sound
It is as if a great brown ground

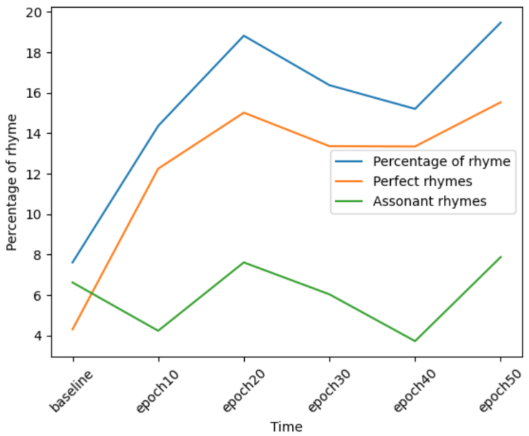

Figure 3: Proportion of perfect and assonant rhymes when training on natural AABB data for 50 epochs.

5.3 Learning from Natural Data

In this experiment, we attempt to teach GPoeT the AABB rhyming pattern using natural rather than synthetic data. We extract from the above-mentioned Chicago Rhyming Poetry Corpus all couplets with consecutive rhyming lines, resulting in a dataset of 2.25 MB of text, mainly with perfect rhymes (75% according to our metric). All other parameters are identical to those of the previous section.

The evolution of the proportions of perfect rhymes and assonant rhymes generated every 10 epochs during training is shown in Figure 3. The proportions are significantly smaller than in the previous experiment, and as the total proportion of rhymes never surpassed 20%, we only represent 50 epochs in the figure. While the model still outperforms the baseline (which has only 7.6% of rhyming verses), it is noticeably less successful than the previous one. It is likely that the smaller amount of data (by a factor of 3) and the larger variety of the vocabulary used by human poets vs. GPT-2 are the main causes of the lower performance.

5.4 Learning the ABAB Pattern

The ABAB rhyming pattern seems more challenging to learn, as line-endings which should rhyme are further apart, separated by one verse. In this experiment, we use our second synthetic dataset, with ABAB quatrains, without separating them into couplets. Quatrains are separated by a blank line. All other parameters are identical to those of the first experiment.

We train the model until the scores stabilize,

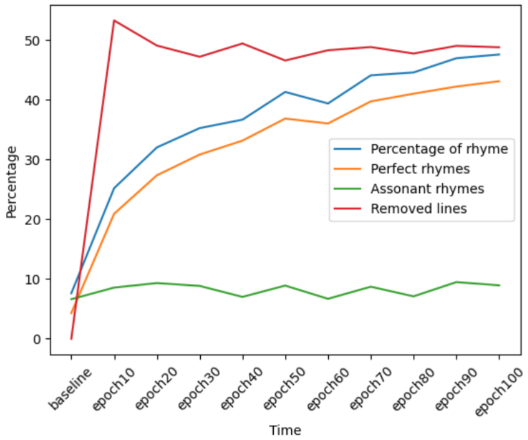

Figure 4: Proportion of perfect and assonant rhymes generated every 10 epochs when training on ABAB synthetic data.

which is around 80 epochs, as shown in Figure 4. The proportion of perfect rhymes rises quickly and converges at around 40%, with a total number of rhyming verses (perfect and assonant) around 45%. Among these, 82.6% are perfect rhymes. As before, to evaluate rhyming, we delete solitary lines, i.e., lines that are not in a quatrain. The proportion of lines retained is 51%, which is much less than above (80%), likely because it is harder to learn to generate a quatrain than a couplet. However, when it generates a full quatrain, the model has clearly learned the ABAB rhyming scheme, although to a lesser extent than the AABB scheme (45% compared to 59%).

6 Related Work

Before the advent of deep neural LMs, various combinations of rule-based approaches and n-gram LMs have been tried. For instance, McGregor et al. (2016) defined a poem generation system which included a phonological model "to impose a sense of prosody" but not dealing with rhymes. In fact, rhyming was not considered the most urgent problem to solve as LMs were struggling with fluency and, especially, meaning.

Large neural LMs have brought high expectations regarding their capacities to generate structured texts such as poems, and clearly improved fluency for high-resource languages. Poem generation with GPT-2 (Radford et al., 2019) was discussed, for instance, by Branwen and Presser (2019) in a blog entry shortly after the model was made available. More recently, ChatGPT (OpenAI, 2022) has tremendously improved the quality and relevance of generated text. However, anecdotal evidence shows that it cannot reliably generate a given rhyming pattern.[11] GPT-4 (OpenAI, 2023), an even larger LM, is likely to improve this capability, as initial analyses seem to show (Bubeck et al., 2023, Sections 1.1 and 6.2).

LMs based on recurrent neural networks (RNNs) were trained by Hopkins and Kiela (2017) on 1.5 MB of English sonnets, first with a single phonetic model and HMM-based phonetic-to-orthographic transliteration, and then with decoupled models for content vs. form. Rhymes from the first model were exemplified, but not evaluated, while the second approach targeted only rhythm, but not rhyme.

'Deep-speare' (Lau et al., 2018) is a LSTM-based system trained on sonnets (2,685 poems), which includes a dedicated orthographic rhyming model, distinct from the LM and from the rhythmic model. The model learns to distinguish rhyming from non-rhyming words in non-annotated quatrains, and during generation it is applied like our rule-based algorithm to select line endings that rhyme. Evaluation is done over word pairs from the CMU pronunciation dictionary, using rules similar to ours to determine ground truth; on this task, their system reaches 0.91 F_1-score.

Wöckener et al. (2021) trained an end-to-end unidirectional word-level RNN on quatrains from the Chicago Rhyming Poetry Corpus. The RNN obeys user-specified constraints such as rhyme, alliteration, sentiment, text length, and time period. These are represented as a feature vector c and concatenated to every input representation to compute $P(w_t|w_0^{t-1}, c)$. Evaluation of rhymes is done with a supervised model (Haider and Kuhn, 2018). They also attempt to fine-tune GPT-2 on pseudo-quatrains from Project Gutenberg, but find that the model does not learn the relevant patterns. They observe an accuracy of 7.5% for rhyming, when compared with a random baseline of 4.2%.

For Chinese, one of the earliest systems using

[11] When asked 'What are the possible rhyming patterns?', ChatGPT enumerates several patterns with definitions and examples, but with factual mistakes such as "ABAB: In this pattern, each line rhymes with the line that comes after it." Moreover, the example generated by ChatGPT for the ABAB pattern is an ABCB stanza. When prompted to "write one quatrain about the ocean, make the first verse rhyme with the third one, and the second with the fourth", ChatGPT generates three fluent quatrains, but with incorrect rhyming patterns (AAAB, CCDE, and FFAA). Moreover, ChatGPT seems unable to reliably generate verses (or even plain sentences) with a fixed number of syllables or words larger than about 7.

RNNs was proposed by Zhang and Lapata (2014), starting from user-provided keywords and generating a quatrain line-by-line, with pre-defined line lengths and tonal patterns. Rhyming is only enforced between the second and fourth lines, simply by disallowing the decoder to select ending characters that do not rhyme. The constraints are similar to the method of Yan et al. (2013) who used a generative summarization approach. Li et al. (2018) built a Chinese poem generator using a variational encoder and adversarial training, starting from a title. Poems were evaluated for topic consistency, fluency, and meaning, but not explicitly for rhyming. Yang et al. (2019) studied the problem of generating a poem from prose and compared LSTM to Transformer models, but did not model explicitly rhymes, nor evaluated them.

PoeTryMe is a rule-based interactive poem generation system initially designed for Portuguese and later extended to Spanish and English (Gonçalo Oliveira, 2017). In the interactive version,[12] assistance is provided to users for selecting end-of-line words that rhyme, through a dictionary. In the standalone Twitter bot (`@poetartificial`), candidates which happen to contain rhyming lines more than others are rewarded. Poem Machine (Hämäläinen, 2018) is an assistant for Finnish, which provides help for rhyming via a phonetic dictionary, but does not select rhyming words automatically. Our own CR-PO system for French (Popescu-Belis et al., 2022), combined a general LM with topic and emotion-specific LMs, and with rules for constraining form and rhymes (the latter are used in this paper).

Hafez was one of the first systems to combine interaction and deep neural LMs (Ghazvininejad et al., 2016, 2017). The system gets the desired features from the user, including keywords and sentiment, transforms them into transducers, and uses a RNN filtered by these transducers to generate a quatrain. Rhyming words are generated early in the process, using word2vec similarity and a phonetic representation, typically in an ABAB pattern, and afterwards they constrain the generation of the poem. Henceforth, rhyming is always ensured.

Van de Cruys (2019, 2020) proposed a RNN encoder-decoder architecture with attention, with GRUs, for English and French poems. The model is trained to generate a line of poetry given the preceding one, with a decoder part that models the new line in reverse order. The advantage of starting from the last word, as for Hafez, is that it can be sampled with a probability distribution that incorporates rhyming constraints, using a rhyming dictionary similar to ours, with an additional bias to avoid repeating the consonant group preceding the final vowel [+ consonant]. In the experiments, the ABAB CDCD pattern is always used. Human judges ranked a set of 40 generated poems almost as high as human ones on several parameters. No scores are provided for rhyming alone, likely because it is nearly perfect given the architecture, but the rhyming component improved scores of 'poeticness' and human-likeliness.

PoeLM (Ormazabal et al., 2022) uses a decoder (GPT-style with 350M parameters) to learn rhythm and syllables from a large corpus of prose in Spanish and Basque. Input text is segmented into phrases, and for each set of phrases of a sentence a set of tags is prepended to the sentence, e.g. `<LEN:11><END:ura>` for an 11-syllable phrase finishing with '-ura'. PoeLM learns these control tags and can leverage them to generate lines of poetry of desired length and endings. However, as the model does not learn rhyming rules (i.e. identity of syllables) but only identifies actual syllables, poem generation must start by specifying exactly the ending syllables of each line. Evaluation is done by completing the initial line of human poems with PoeLM, and then asking human judges which version they prefer.

ByGPT5 (Belouadi and Eger, 2022) is a character-level Transformer-based decoder, with generation conditioned on rhyme, meter, and alliteration. The model is initialized on the decoder of ByT5 (Xue et al., 2022), trained on large amounts of data, and then fine-tuned on a machine-labeled corpus of pseudo-quatrains in English and German, separately. Meter and rhyme are evaluated with classifiers trained on labeled data. Overall, according to automatic and human measures, ByGPT5 produces better results than ByT5 and subword-level models such as GPT-2 and mT5.

7 Discussion and Conclusion

The rhyme-generating LM presented here, GPoeT, is intended for integration in our interactive poem generation system (Popescu-Belis et al., 2022). While the experiments above show that rhyming patterns can be learned from synthetic data, several issues remain to be solved in future studies.

[12] See poetryme.dei.uc.pt.

Our rule-based rhyming algorithm operates on a poem already generated by a LM with several other parameters as input, e.g. a title or first verse, a desired theme or emotion, and a poetical form (such as a sonnet). We must now integrate GPoeT in this pipeline, and ensure that the generated rhymes are not altered by the other constrains of the system. Moreover, we must ensure that the lexical diversity of GPoeT is not reduced by its training on synthetic data.

We intend to address the problem of generating a desired form using the rule-based algorithm presented in Section 3.2, which takes advantage of a maximum length for the LM decoder. It may seem straightforward to replace GPT-2 with GPoeT in this algorithm, in order to obtain rhyming lines of a desired length, but our initial experiments have shown that rhymes are less satisfactory when the desired length is very different from the synthetic data GPoeT was trained on.

Moreover, while our rule-based rhyme generator can be easily adapted to any rhyming pattern, this is not yet the case for GPoeT, which is trained on one pattern at a time in our proof-of-concept. The solution lies probably in using a labeling system to indicate which lines must rhyme, and then training a GPoeT model to learn the effects of labels rather than a single rhyming pattern, in the style of the CTRL model (Keskar et al., 2019).

In this paper, we demonstrated that rhyming is learnable with LMs that can be efficiently fine-tuned and queried with very moderate computing requirements. The key to effective fine-tuning is the use of synthetic data, which we showed how to generate in much larger amounts than what human poets have ever written. However, not all rhyming patterns are learned equally well: a pattern that exhibits longer-term dependencies such as ABAB is harder to learn than a more local one such as AABB. Overall, LMs that are able to deal with rhyme, and later with form, are part of our ongoing effort to design an interactive poetry generator, with the aim of enhancing (but not replacing) human creativity.

Limitations

The technical limitations of this study were discussed to some extent at the beginning of the conclusion (Section 7) and will result in future investigations regarding the generation of specific poetic forms (and line lengths) and on-the-fly selection of rhyming patterns. Our study relies on a phonetic dictionary of English, along with rhyming definitions related to the English-speaking culture: these must be redesigned when porting the system to a new language. The results may have been limited by the use of a rather small LM and reduced computing time, but this also has the advantage of reduced power consumption, and makes it possible to demonstrate the system on a standalone portable workstation.

Ethics Statement

The ethical issues broadly related to the of LMs for text generation also apply to poetry: the generation of offensive content, the reproduction of unethical stereotypes learned from the data, and the substitution of human creativity by machines. While we do not have quick answers to these large societal questions, we observed that due to its training on classic poetry, GPoeT is not likely to generate offensive content (for instance, filtering out bad words has proven unnecessary). Our goal is not the fully-autonomous generation of poems, but co-creation of poetry with human users, who have to steer the system towards a desired form and topic. Our approach is intended to stimulate human creativity, not to replace it.

Acknowledgments

We would like to thank the Swiss National Science Foundation (SNSF) for its support through the Digital Lyric (n. 184330) and the DOMAT projects (n. 175693). We are grateful for the support received from the HES-SO through the PhD fund (AGP n. 107554 and 119509) and from the Institute for ICT at HEIG-VD. We thank EPFL Professors Martin Jaggi and Nicolas Flammarion for the opportunity to conduct a project in their CS-433 machine learning course, and the three anonymous LaTeCH-CLfL reviewers for their helpful suggestions.

References

Àlex R. Atrio and Andrei Popescu-Belis. 2022. On the interaction of regularization factors in low-resource neural machine translation. In *Proceedings of the 23rd Annual Conference of the European Association for Machine Translation (EAMT)*, pages 111–120, Ghent, Belgium. European Association for Machine Translation.

Jonas Belouadi and Steffen Eger. 2022. ByGPT5: End-to-end style-conditioned poetry generation with token-free language models. *arXiv preprint arXiv:2212.10474*.

Gwern Branwen and Shawn Presser. 2019. GPT-2 neural network poetry. *Demo Tutorial*.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv preprint arXiv:2303.12712*.

Marjan Ghazvininejad, Xing Shi, Yejin Choi, and Kevin Knight. 2016. Generating topical poetry. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1183–1191, Austin, Texas. Association for Computational Linguistics.

Marjan Ghazvininejad, Xing Shi, Jay Priyadarshi, and Kevin Knight. 2017. Hafez: an interactive poetry generation system. In *Proceedings of ACL 2017, System Demonstrations*, pages 43–48, Vancouver, Canada. Association for Computational Linguistics.

Hugo Gonçalo Oliveira. 2017. O Poeta Artificial 2.0: Increasing meaningfulness in a poetry generation Twitter bot. In *Proceedings of the Workshop on Computational Creativity in Natural Language Generation (CC-NLG 2017)*, pages 11–20, Santiago de Compostela, Spain. Association for Computational Linguistics.

Thomas Haider and Jonas Kuhn. 2018. Supervised rhyme detection with Siamese recurrent networks. In *Proceedings of the Second Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, pages 81–86, Santa Fe, New Mexico. Association for Computational Linguistics.

Mika Hämäläinen. 2018. Poem Machine – a co-creative NLG web application for poem writing. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 195–196, Tilburg University, The Netherlands. Association for Computational Linguistics.

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*, pages 168–178, Vancouver, Canada. Association for Computational Linguistics.

Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. 2019. CTRL: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*.

Jey Han Lau, Trevor Cohn, Timothy Baldwin, Julian Brooke, and Adam Hammond. 2018. Deep-speare: A joint neural model of poetic language, meter and rhyme. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 1948–1958, Melbourne, Australia. Association for Computational Linguistics.

Juntao Li, Yan Song, Haisong Zhang, Dongmin Chen, Shuming Shi, Dongyan Zhao, and Rui Yan. 2018. Generating classical Chinese poems via conditional variational autoencoder and adversarial training. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3890–3900, Brussels, Belgium. Association for Computational Linguistics.

Stephen McGregor, Matthew Purver, and Geraint Wiggins. 2016. Process based evaluation of computer generated poetry. In *Proceedings of the INLG 2016 Workshop on Computational Creativity in Natural Language Generation*, pages 51–60, Edinburgh, UK. Association for Computational Linguistics.

OpenAI. 2022. ChatGPT: Optimizing language models for dialogue. *OpenAI Blog*.

OpenAI. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Aitor Ormazabal, Mikel Artetxe, Manex Agirrezabal, Aitor Soroa, and Eneko Agirre. 2022. PoeLM: A meter- and rhyme-controllable language model for unsupervised poetry generation. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 3655–3670, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Andrei Popescu-Belis, Àlex Atrio, Valentin Minder, Aris Xanthos, Gabriel Luthier, Simon Mattei, and Antonio Rodriguez. 2022. Constrained language models for interactive poem generation. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 3519–3529, Marseille, France. European Language Resources Association.

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

David Uthus, Maria Voitovich, and R.J. Mical. 2022. Augmenting poetry composition with Verse by Verse. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track*, pages 18–26, Hybrid: Seattle, Washington + Online. Association for Computational Linguistics.

Tim Van de Cruys. 2019. La génération automatique de poésie en français. In *Actes de la Conférence sur le Traitement Automatique des Langues Naturelles (TALN) PFIA 2019. Volume I : Articles longs*, pages 113–126, Toulouse, France. ATALA.

Tim Van de Cruys. 2020. Automatic poetry generation from prosaic text. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2471–2480, Online. Association for Computational Linguistics.

Jörg Wöckener, Thomas Haider, Tristan Miller, The-Khang Nguyen, Thanh Tung Linh Nguyen, Minh Vu Pham, Jonas Belouadi, and Steffen Eger. 2021. End-to-end style-conditioned poetry generation: What does it take to learn from examples alone? In *Proceedings of the 5th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, pages 57–66, Punta Cana, Dominican Republic (online). Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's Transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306.

Rui Yan, Han Jiang, Mirella Lapata, Shou-De Lin, Xueqiang Lv, and Xiaoming Li. 2013. i, Poet: Automatic Chinese poetry composition through a generative summarization framework under constrained optimization. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2197–2203. AAAI Press.

Zhichao Yang, Pengshan Cai, Yansong Feng, Fei Li, Weijiang Feng, Elena Suet-Ying Chiu, and Hong Yu. 2019. Generating classical Chinese poems from vernacular Chinese. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6155–6164, Hong Kong, China. Association for Computational Linguistics.

Xingxing Zhang and Mirella Lapata. 2014. Chinese poetry generation with recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 670–680, Doha, Qatar. Association for Computational Linguistics.