GPT Czech Poet: Generation of Czech Poetic Strophes with Language Models

Anonymous ACL submission

Abstract

High-quality automated poetry generation systems are currently only available for a small subset of languages. We introduce a new model for generating poetry in Czech, a heavily inflected Slavic language with rather regular orthography and prosody. We find that appropriate tokenization is crucial, showing that tokenization methods based on syllables or individual characters instead of subwords prove superior in generating poetic strophes. We also demonstrate that guiding the generation process by explicitly specifying strophe parameters within the poem text can improve the effectiveness of the model. We further enhance the results by introducing Forced Generation, adding explicit specifications of meter and verse parameters at inference time based on the already generated text. We evaluate a range of setups, showing that our proposed approach achieves high accuracies in several aspects of formal quality of the generated poems.

1 Introduction

End-to-end pre-trained language models, such as GPT-2 (Radford et al., 2019) or Llama-2 (Touvron et al., 2023), have gained immense popularity for fine-tuning on various downstream tasks. The emergence of Large Language Models (LLMs), notably those fine-tuned on dialog data and open-domain communication such as Orca (Mukherjee et al., 2023) or ChatGPT/GPT-4 (OpenAI, 2023), has introduced a paradigm shift in model adaptation, moving away from traditional fine-tuning towards a more prompt-centric approach.

However, despite their versatility, open-domain models may face a potential drawback when applied to languages and tasks less prevalent in the training data (Liu et al., 2023). The Czech language and Czech poetry present such a scenario, where the models, lacking sufficient exposure during training, struggle to adhere to the structural nuances of strophes and the associated parameters, resulting in sub-optimal performance on these specific tasks. Therefore, we resort to the more traditional practice of fine-tuning GPT base models.

The Czech language also differs in several important characteristics from other usually studied languages, most notably by its rich inflection but rather regular orthography and prosody, which motivates the approach we take in this work.

We draw inspiration from treating text as a sequence of syllables (Oncevay and Rojas, 2020). Our primary focus lies not in the semantic intricacies of the text, a domain where models with standard tokenizers like BPE (Wang et al., 2019) excel, but rather in the phonetic aspects and the adherence to meter, which are paramount for our task. Syllabic modeling proves particularly advantageous in generating neologisms, common in poetry to maintain prescribed rhyme scheme and meter.

In pursuit of this, we have delved into tokenizer-free models (Xue et al., 2022), offering maximal flexibility in constructing neologisms and pairing characters to align with stipulated strophe parameters. This approach, already demonstrated to be effective in poetry generation by the byGPT5 system (Belouadi and Eger, 2023), showcased proficiency in both rhyme scheme and meter adherence.

We also experiment with several ways of guiding the generation process by interleaving explicit annotations with the strophe text.

Tvá loď jde po vy-so-kém mo-ři, ⏑ – – – ⏑ ⏑ ⏑ – ⏑	**A**	*iamb*
v ně brá-zdu ja-ko stří-bro re-je, ⏑ – ⏑ – ⏑ – ⏑ – ⏑	**B**	*iamb*
svou pří-du v mod-ré vl-ny no-ří ⏑ – ⏑ – ⏑ – ⏑ – ⏑	**A**	*iamb*
a bok svůj pěn-né do pe-ře-je. ⏑ – ⏑ – ⏑ – ⏑ ⏑ ⏑	**B**	*iamb*

Table 1: An **ABAB** strophe with meter annotation.

2 Parameters of Poetry

In poetic strophes, there are two main parameters that govern their structure: rhyme and meter (even though many strophes are crafted without adhering to rhyme or are constructed in free verse). While the rhyme scheme applies to the entire strophe, the meter may vary from verse to verse. Consequently, in our analysis, we meticulously annotate the meter for each individual verse.

2.1 Rhyme

Utilizing the standard approach, we designate the rhyme scheme with capital letters, such as **ABAB**, where each character denotes an individual verse in the strophe, also allowing **X** for non-rhyming verses. We include configurations of both 4 and 6 lines. The rhyming scheme thus can be e.g. **AABBCC**, where each verse has a corresponding rhyming pair, as well as e.g. **XAXA**, where only the second verse rhymes with the fourth.

2.2 Meter

We considered the following meter types that occur in our dataset (labelled with one-letter labels):

iamb (J) First syllable is short and unstressed, second is long and stressed. E.g. 'attempt' => 'at-tempt', stress is on second syllable 'tempt'.

trochee (T) Reverse of iamb, first syllable is stressed, second is unstressed. E.g. 'double' => 'dou-ble' with stress on first syllable.

dactyl (D) Three part meter with stress on first long syllable. Next two syllables are short and unstressed. E.g. 'poetry' => 'po-et-ry' with stress on first syllable.

amphibrach (A) Three part meter with stress on second syllable. E.g. 'the scenes of', where stress is placed on the word 'scenes'.

dactylotrochee (X) Combination of dactyl and trochee.

dactylotrochee with anacrusis (Y) Anacrusis is a set of unstressed syllables preceding the first stressed dactylotrochee syllable.

hexameter (H) Non-rhyming verse with 6 parts.

pentameter (P) Non-rhyming verse with 5 parts.

free verse (N) Does not pertain to any meter.

See Figure 1 for an example of a strophe with the **ABAB** rhyme scheme and *iamb* meter for each verse. To illustrate how each verse adheres to the iambic meter, we mark unstressed syllables with "⌣" and stressed syllables with "-".

3 Dataset

We opted for the Corpus of Czech Verse (Plecháč and Kolár, 2015), curated by the Institute of Czech Literature of the Czech Academy of Sciences.[1] This corpus comprises 1,305 volumes of poetry, each annotated for poetic meters, rhymes, phonetic transcription, word tokenization, lemmatization, and morphological tagging. The annotation is semi-automatic and can thus contain errors; e.g. meter annotation has an estimated accuracy of 95.3% (Plecháč, 2016). The metadata include information such as the author name, book editors, and the publication years of the book.

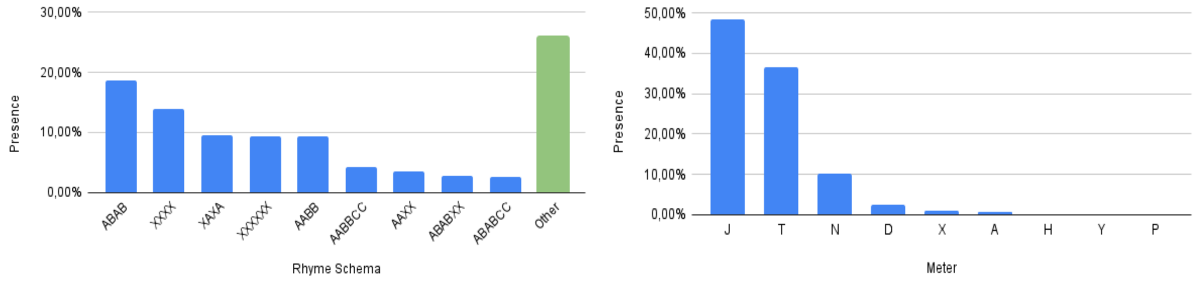
3.1 Dataset Preprocessing

The utilized corpus lacks direct specification of rhyme schemes, instead providing information on whether two or more verses rhyme or if a verse is non-rhyming. Consequently, we transformed this information into standardized rhyme schemes such as **AABB, AABBCC**, as discussed earlier. Given that the metadata lacks details about the type of poetry (Lyric, Narrative) or the specific style in which a poem was composed, we inferred that the publication year of the book containing the poem serves as the most indicative feature. However, as Language Models struggle with numerical data and benefit from fine-tuning for improved comprehension (Spithourakis and Riedel, 2018), we bucketized the publishing year data into 20-year periods to better categorize poems into distinct styles. Some poems lacked information about their publication year, and for these instances, we introduced the category NaN to encompass such examples.

3.2 Dataset Makeup

To gain a more comprehensive understanding of potential biases in our model, it was crucial to scrutinize the composition of the processed data. The combined corpus encompasses 2,310,917 verses, forming 374,537 strophes, which collectively constitute 66,428 poems. We split the dataset into a train set (95%) and a test set (5%).

[1] https://github.com/versotym/corpusCzechVerse



(a) Top 10 Rhyme schemes presence

(b) Meter presence

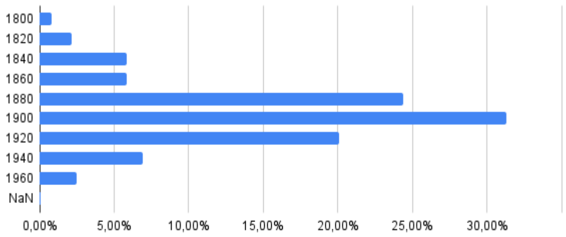


Figure 2: Year regions presence

Rhyme schemes Our processing identified 218 different schemes (primarily due to our leniency towards non-rhyming verses), with a very uneven distribution. Figure 1a depicts the 10 most frequent rhyme schemes, which together constitute 74% of the dataset. Conversely, we identified 149 distinct rhyme schemes with a presence below 0.05% each (fewer than 200 strophes) in our corpus, thus probably constituting noise rather than meaningful patterns that our model could learn from.

Meter We observe a modest variety with only 9 distinct types of meter (8 metric and 1 free verse). However, as illustrated in Figure 1b, over 85% of all verses pertain to either iamb (J) or trochee (T), whereas the least frequent meter types (H, Y, P) each individually constitute less than 0.2% of the data. Therefore, in the absence of specific instructions, our model is likely to predominantly generate J and T verses.

Year of poem publication Figure 2 illustrates a more even distribution across all categories than for rhyme schemes and meters. Only NaN exhibits a presence below 0.5%, while 6 out of the 10 defined regions have a presence exceeding 5%.

4 Data Format

Standard language modelling is done on the plain text. However, for poetry modelling, previous works have demonstrated strong benefits of explicitly encoding various properties within the text by using annotations via functional tokens interleaved with the actual language tokens. We therefore explore three variants of specifying strophe and verse parameters.

BASIC Our initial method, as previously explored in the ByGPT5 article (Belouadi and Eger, 2023), involves adding the rhyme scheme, theme (i.e. publishing year), and the most prevalent meter as the first line, while the subsequent lines contain the strophe in plain text; see the example in Figure 3.

ABAB # 1900 # J
Tvá lod' jde po vysokém moři,
v ně brázdu jako stříbro reje,
svou přídu v modré vlny noří
a bok svůj pěnné do peřeje.

Figure 3: Example of a strophe using the BASIC model input format.

VERSE_PAR While the initial approach is promising, insights from the GPoet-2 article (Lo et al., 2022) indicate that relying solely on raw attention may be insufficient, necessitating reverse modeling to achieve rhyming verses. In response to this, we considered the inclusion of a set of verse parameters, **syllable line length** and **ending syllable**, as a prefix to each line, to provide more guidance to the attention mechanism in individual verses. This modification is reflected in the example in Figure 4.

ABAB # 1900 # J
9 # ři # Tvá lod' jde po vysokém moři,
9 # je # v ně brázdu jako stříbro reje,
9 # ří # svou přídu v modré vlny noří
9 # je # a bok svůj pěnné do peřeje.

Figure 4: Example of a strophe using the VERSE_PAR model input format with verse parameters.

METER_VERSE Building upon our prior considerations, and given the availability of data for the meter of each individual verse, we recognize the potential value in incorporating meter information for each verse individually instead of the full strophe. This additional input, which can vary between sets of rhyming verses (e.g., from iamb to trochee), provides enhanced guidance to the attention mechanism, particularly in achieving a clear separation of non-rhyming verses. The resulting input scheme is illustrated in Figure 5.

ABAB # 1900
J # 9 # ři # Tvá loď jde po vysokém moři,
J # 9 # je # v ně brázdu jako stříbro reje,
J # 9 # ří # svou přídu v modré vlny noří
J # 9 # je # a bok svůj pěnné do peřeje.

Figure 5: Example of a strophe using the METER_VERSE model input format with meter as verse parameter.

5 Tokenization

We recognize tokenization as a critical element in our task, given our emphasis on formal aspects (rhyming, meter) rather than meaning, as well as our explicit inclusion of functional tokens specifying desired properties (rhyming, meter, year) interleaved with actual language tokens. We embarked on a series of experiments to address the following objectives:

- Distinguish between actual language tokens and functional tokens.
- Segment words into tokens that aid in guiding meter and inflection.
- Facilitate the swapping of small chunks to encourage fitting the formal requirements and the generation of neologisms.

The standard approach in current NLP is subword tokenization, such as BPE (Sennrich et al., 2016). Given the nature of the Czech language with its reliance on inflection, our focus on formal properties, and the incorporation of neologisms in poetry, particularly for rhyming purposes, we also drew inspiration from approaches involving the separation of words into syllables (Oncevay and Rojas, 2020) or even individual characters (Xue et al., 2022).

Therefore, we experiment with the following four tokenization approaches:

BASE The original tokenizer of the `czech-gpt2-oscar` model (Chaloupský, 2022) which we use.

OUR A BPE tokenizer trained on our dataset.

SYLLABLE Splitting the text into syllables, using the *Sekacek* tool (Macháček, 2014).[2]

UNICODE Splitting the text into individual characters.

The benefit of training a standard BPE tokenizer on our dataset is that it can learn to keep functional annotations as single tokens, as shown in Figure 6.[3]

INPUT:	# ABAB # 1900
BASE:	[#] [AB] [AB] [#] [1900]
OUR:	[#] [ABAB] [#] [1900]
SYLL.:	[#] [ABAB] [#] [1900]
UNIC.:	[#][ ][A][B][A][B][ ][#][ ][1][9][0][0]

Figure 6: Tokenization of strophe parameters.

Obviously, SYLLABLE and UNICODE encode sequences into larger amounts of shorter tokens; see Figure 7. This allows the model to make fine generation decisions with a higher granularity, so that it can better fit the prescribed formal properties (meter, rhyme). It also makes production of nealogisms easier. However, as mentioned by Belouadi and Eger (2023), the time required for model training and inference increases accordingly.

INPUT:	a v duchu
BASE:	[a] [v] [duchu]
OUR:	[a] [v] [duchu]
SYLLABLE:	[a] [v] [duch] [u]
UNICODE:	[a] [ ] [ ] [v] [ ] [d] [u] [c] [h] [u]

Figure 7: Tokenization of verse text.

6 Training the Models

As our base model, we have selected `czech-gpt2-oscar` by Chaloupský (2022),[4] a GPT-2-small model (Radford et al., 2019) trained on the Czech part of the OSCAR dataset (Suárez et al., 2020).

[2] `https://github.com/Gldkslfmsd/sekacek`
[3] Of course, this is only effective for annotations that are sufficiently frequent in our dataset.
[4] `https://huggingface.co/lchaloupsky/czech-gpt2-oscar`

Tokenizer	Model Parameters
BASE	137M
OUR	137M
SYLLABLE	105M
UNICODE	86M

Table 2: Sizes of the fine-tuned models, depending on the tokenization approach.

We then fine-tune the model on our dataset, using one of the three data formats (Section 4) and one of the four tokenizers (Section 5).

We explore two different approaches of training the model for the selected data format. We either simply train the model using only the selected data format, or we first pre-train the model using the METER_VERSE format, and then fine-tune it using the BASIC or VERSE_PAR format; the motivation for this approach is that each of these formats can be regarded as a subset of the METER_VERSE format.

For our model to accept the format of strophe and to follow the parameters of strophe and verses in it, we have, instead of using secondary tasks, utilized only attention, as recommended by Vaswani et al. (2017). For our loss computation, we employ the conventional Cross Entropy Loss, with our input serving as labels as well. Given the GPT-based nature of our model, we refrain from employing input masking, as the preferred training method for GPT-2 involves next word prediction.

For using our custom tokenizers, we have followed the model recycling approach of de Vries and Nissim (2021), which utilizes overlap in current and target vocabularies to jump-start the model by keeping large parts of the embedding matrix.

The sizes of the resulting fine-tuned models can be seen in Table 2. As SYLLABLE and UNICODE tokenizers have smaller vocabularies, the resulting models are smaller; on the other hand, the data format has no effect on the model size.

7 Text Generation

To further enhance the model's proficiency in adhering to strophe and verse parameters at inference, we propose an alternative approach to the standard text generation method.

Basic Decoding The prompt consists of the first line which specifies the strophe parameters. Then, generation proceeds token by token until the end-of-sequence token is generated.

Forced Generation This iterative method involves examining an already accepted rhyme scheme and compelling verse parameters for lines intended to rhyme. After generating each verse, the generation process stops, and if the next verse to be generated should rhyme with an already generated verse, then the verse parameters are copied (forced) as the prefix for the next line before resuming the generation process, as illustrated in Figure 8. More formally, if the model has already generated meter *X*, syllable length *Y* and ending syllable *Z* as annotations for a verse connected to character **A** in the rhyme scheme, all other verses linked to character **A** will be prompted with verse parameters *X* # *Y* # *Z* #. Obviously, this approach is only applicable for VERSE_PAR and METER_VERSE input formats.

AABB # 1900
T # 8 # ání # A když přijde z nenadání,
T # 8 # ání # ...

Figure 8: Forced Generation. According to the AABB rhyme scheme, the second verse should rhyme with the first verse. Thus, after generating the first verse, the verse parameters for the second verse (underlined) are forced, i.e. copied from the first verse (**in bold**).

We have also experimented with beam-search and top-k sampling. For the UNICODE tokenizer, this led to better results, while other models remained unaffected. Consequently, we will report results using the best setup for each model.

8 Validators

Comprehensive automated quality evaluation of text generation is hard. In our setting, we have decided to focus on a narrower subtask, mostly evaluating formal quality of the generated poetry. Rule-based approaches exist (Plecháč, 2018), but given the large annotated dataset at our disposal, we can train validator models directly on the dataset. Specifically, we train classifiers that label strophes with the **rhyme scheme**, **meter**, and **year**. We can then simply evaluate whether the predicted value matches the value specified on the input.

The general approach we take is to train a softmax classifier attached to the class token representation in a masked language model; we use either RoBERTa (Liu et al., 2019), or its Czech version, RobeCzech (Straka et al., 2021).

Base model	Input type	Accuracy
robeczech-base	Syllable	**97.17** %
robeczech-base	Raw	72.77 %
roberta-base	Syllable	**96.66** %
roberta-base	Raw	79.91 %
Baseline	NA	*18.65 %*

Table 3: Rhyme scheme prediction validator.

Base model	Input type	Accuracy
robeczech-base	Syllable	**95.60** %
robeczech-base	Raw	84.36 %
roberta-base	Syllable	**95.60** %
roberta-base	Raw	90.90 %
Baseline	NA	*48.52 %*
Upper bound	NA	*95.30 %*

Table 4: Meter prediction validator.

8.1 Validator Input Preprocessing

As syllables are useful text units when concerned with formal properties of poetry, we again experiment with splitting the input into syllables before feeding it into the RoBERTa/RobeCzech model.

This approach simplifies the tasks of rhyme and meter validators, as they no longer need to guess word partitioning. Their focus is now solely on learning syllabic rhyming patterns and stress patterns associated with syllables.

However, the effectiveness of syllabification for the year validator is uncertain. Understanding themes requires both grasping the employed metrical and rhyming structures, where syllabification helps, as well as discerning the semantic meaning, where syllabification causes a partial disruption.

8.2 Validators Accuracies

Using the train and test parts of the dataset, we train and evaluate validators for rhyme scheme prediction (Table 3), meter prediction (Table 4) and publishing year prediction (Table 5). We also report the **Baseline** as the most common class, and for meter, we have included an **Upper bound** based on the accuracy of the semi-automatic annotation in the dataset (Plecháč, 2016).

Syllabification Pre-splitting the input into syllables significantly aids the validators in classifying syllable-based parameters, i.e. meter and rhyme scheme, but seems to be irrelevant or even harmful for the year classification. This aligns with our expectations, as the year of publishing is more closely tied to the subject of the poem, a facet disrupted by the syllabification process.

Base model	Input type	Accuracy
robeczech-base	Syllable	**58.93** %
robeczech-base	Raw	**58.86** %
roberta-base	Syllable	41.72 %
roberta-base	Raw	47.79 %
Baseline	NA	*31.33 %*

Table 5: Year of publishing prediction validator.

Rhyme scheme and meter prediction The validators on syllabified input achieve very high accuracies, reaching or approaching the maximum accuracies achievable on the dataset, as the semi-automated annotation of the dataset is not perfect and contains errors. The accuracies of RobeCzech are slightly higher than RoBERTa or identical.

Year prediction Using RobeCzech leads to significantly higher accuracies than using RoBERTa. We believe this is because this task also requires understanding the semantics of the text, whereas the other tasks focus on the formal properties of the text, and thus the model pre-trained on Czech data has a significant advantage. Still, all the accuracies on this task are rather low, and we do not deem them sufficient for using this validator to reliably evaluate the results of poetry generation.

Token granularity In the context of rhyme scheme and meter, we have observed that the effect of syllabification is less pronounced for RoBERTa than for RobeCzech. We posit that this is because RoBERTa is not pre-trained on Czech texts and thus its subword tokenization needs to split the text into shorter tokens to represent Czech words.

Tokenizer	Chars per token
roberta-base	1.5
robeczech-base	2.7

Table 6: Tokenizer influence on token granularity

We evaluated the model tokenizers by analyzing 10,000 verses and calculating the average number of characters per token. As showcased in Table 6, RoBERTa already tokenizes the text more granularly, resulting in further syllabification having a weaker effect than in the case of RobeCzech.

9 Model Validation

Through our validators, we can evaluate the poetry generation model's adherence to the rhyme scheme and meter. In addition to these metrics, we also assess conformity to the number of syllables and the ending syllable for each verse as generated (or forced) in the prefix annotation at the start of the line. We also measure the uniqueness of the generated syllables as an indicator for non-repetitiveness. Altogether, we compute these characteristics:

Num Syl Proportion of verses with number of syllables matching the prefix annotation.

End acc Proportion of verses with ending syllable matching the prefix annotation.

Unique Ratio of unique syllables among all syllables in the strophe; the optimal value here is not 100%, but rather the value observed on the true data in the dataset (87.90%).

Rhyme acc Proportion of strophes with rhyme scheme matching the first line annotation.

Meter acc Proportion of strophes with the meter of all verses matching the annotation.

We use the annotations of the strophes in the test part of our dataset as inputs (as in Figure 9), and evaluate the generated outputs (now disregarding the actual texts of the strophes in the test dataset).

AXAX # 1880
J # ...

Figure 9: Example of an input prompt using METER_VERSE format.

9.1 Influence of Data Format

We first evaluate the effect of the data format (Section 4), while using the BASE tokenizer and Basic text generation.

The model was either trained using only the selected data format for 8 epochs, or it was first pretrained using METER_VERSE format for 8 epochs and then fine-tuned for further 4 epochs using the selected format.

Data Format	Pre-train	Rhyme acc	Meter acc
BASIC	False	35.44 %	84.53 %
BASIC	True	57.32 %	85.37 %
VERSE_PAR	False	48.22 %	85.06 %
VERSE_PAR	True	**66.68 %**	86.28 %
METER_VERSE	NA	66.50 %	**87.59 %**

Table 7: Influence of Data Format on accuracy.

Table 7 demonstrates that incorporating the individual verse parameters using either VERSE_PAR or METER_VERSE format significantly contributes to the model performance, particularly in terms of adhering to the rhyme scheme. The inclusion of more detailed meter parameters in METER_VERSE scheme further enhances the ability of the model to follow the correctly meter.

Furthermore, the performance with both BASIC and VERSE_PAR formats improves considerably when the model is first pretrained using the METER_VERSE format.

9.2 Final Validation

Finally, we train four models, exploring all the presented tokenizers (BASE, OUR, SYLLABLE, UNICODE), using the METER_VERSE data format, and training for 16 epochs. We generate strophes using either Basic Decoding or Forced Generation.

As shown in Table 8, the best results are obtained by using the UNICODE tokenizer and Forced Generation, often surpassing the other setups with a large margin. This underscores the viability of character-level large language models, particularly in morphological and phonetic tasks. For meter accuracy, OUR tokenizer and Basic Decoding perform best; however most of the setups perform quite competitively in this characteristic.

9.3 Validation Results Analysis

Forced Generation Our proposed approach to generation consistently demonstrated the ability to significantly enhance rhyme scheme accuracy while only minimally impacting meter accuracy, number of syllables accuracy, ending syllable accuracy, and unique syllables ratio. We posit that the improvements in rhyme scheme accuracy can be attributed to the fact that Forced Generation constrains the model to generate matching verses with the same ending syllable and length in syllables, both of which play a substantial role in rhyming. This constraint is also the reason behind the usual decrease in meter accuracy and unique syllables ratio. The enforced ending syllable is not unique, and it compels the model to generate proper meter inclusive of it, which, especially with single-syllable unstressed words, can pose a challenge.

OUR tokenizer The performance of OUR tokenizer was the least satisfactory among the consid-

Tokenizer	Generation	Num Syl	End acc	Unique	Rhyme acc	Meter acc
BASE	Basic	92.36 %	96.20 %	86.01 %	66.40 %	87.37 %
BASE	Forced	92.55 %	96.22 %	84.72 %	69.62 %	86.40 %
OUR	Basic	91.63 %	94.64 %	84.76 %	47.56 %	**88.17 %**
OUR	Forced	91.67 %	94.52 %	83.46 %	49.14 %	87.44 %
SYLLABLE	Basic	95.84 %	98.17 %	84.73 %	72.10 %	88.09 %
SYLLABLE	Forced	95.57 %	98.18 %	83.39 %	74.12 %	87.08 %
UNICODE	Basic	91.31 %	92.24 %	89.74 %	68.92 %	83.34 %
UNICODE	Forced	**97.49 %**	**98.94 %**	**87.64 %**	**87.96 %**	86.19 %
Target		*100 %*	*100 %*	*87.90 %*	*100 %*	*100 %*

Table 8: Validation results for the final models.

Tokenizer	Chars per token
BASE	3.37
OUR	3.77
SYLLABLE	2.43
UNICODE	1.00

Table 9: Tokenizer influence on token granularity

Tokenizer	Year accuracy
BASE	**54.70 %**
OUR	51.00 %
SYLLABLE	41.76 %
UNICODE	40.90 %

Table 10: Year accuracy as reported by the validator model. For each tokenizer, we report the best result observed among all investigated configurations. Note that the year validator is highly unreliable.

ered options. We contend that this can be attributed to the fact that OUR tokenizer was trained solely on poetry data, comprising only 2 GB in size. The resulting number of characters per token is excessively large, rendering it less efficient for poetry generation. Unlike SYLLABLE or UNICODE tokenizer, OUR tokenizer lacks the capability for syllable or character substitution. To substantiate this observation, we conducted the same analysis as for validator tokenizers (Section 8.2, Table 6). In Table 9, we can observe that OUR tokenizer encodes 3.77 characters per token, which is the highest value among all tokenizers. This characteristic diminishes flexibility, restricting words to be represented by only 1 token.

9.4 Year Accuracy

Driven by curiosity, we also employed our validator to assess the probable publishing year accuracy, which is our proxy for poetic style; keeping in mind that this validator is highly unreliable as its accuracy is rather low. Our hypothesis was grounded in the belief that OUR tokenizer, with its capacity to tokenize entire words in a single token, might excel in tasks oriented more towards semantic meaning.

The results in Table 10 show that the models trained with subword tokenizers (BASE, OUR) achieve distinctly higher scores, which is in line with our expectations. Yet, contrary to our expectations, OUR tokenizer still lags behind BASE tokenizer; this may be an artifact of the unreliable validator, but it may also be the effect of OUR tokenizer being trained on smaller and specific data, constraining its ability to capture meaning as comprehensively as the more versatile BASE tokenizer.

10 Conclusion

In this work, we proposed and implemented a novel comprehensive approach to poetic strophe generation, focusing on formal qualities of poetry. We trained and evaluated our models using a corpus of Czech poetry.

Our results reveal superior rhyming accuracy of character and syllable tokenization compared to standard subword tokenization methods. Moreover, we highlight the significant performance boost achieved by Forced Generation, which encourages the model to generate formally more coherent strophes. This is particularly evident with character tokenization, where rhyming accuracy increased by 19%. We have also shown that enriching the plain text with interleaved explicit annotations can help to better guide the model.

In future work, we want to expand our generation to full poems with strophes that are thematically and schematically connected.

11 Ethical Considerations

A topic of active discussion is whether it is ethical (or even legal) to use various kinds of data for training large language models, e.g. without explicit consents of the data authors. In our work, we train the language model on a dataset composed exclusively of poems in the public domain (due to the authors having died more than 70 years ago), which we consider to be non-problematic.

The base GPT-2 model, which we further finetune on that dataset, was trained on various kinds of data, including potentially problematic data. However, our approach can be in principle applied to any base model; thus, if there is ever a consensus that it is not ethical to use this base model, our approach can be repeated and reevaluated using any other base model.

It is becoming the norm (and may be soon required by laws, such the EU AI Act) to label automaticall generated works as such, e.g. to avoid unintentional spreading of misinformation. To this end, we make sure to always label all our generated poems as automatically generated.

12 Limitations

As any transformer model, our solution grapples with substantial computational complexity (Vaswani et al., 2017), necessitating the use of powerful GPUs (A40 40GB, A100 40GB, H100 80GB) for effective training.

An inherent challenge arises from the use of multiple tokenization techniques, potentially impacting the scalability of next strophe generation. Notably, the UNICODE tokenizer struggles to retain context across two verses, posing a risk of losing crucial information.

Another issue stems from data distributions, as illustrated in Figures 1a and 1b. If not prompted appropriately, the model defaults to a rhyme scheme of ABAB and a meter of iamb. This default behavior is problematic, particularly considering that the model is likely incapable of generating most of the 218 rhyme schemes appearing in the dataset. Regarding meter, only iamb, trochee, and free-verse are reliably generated, with the remaining 6 typically defaulting to iamb.

With our inability to observe if year of publishing is followed (Table 5), it remains uncertain whether the model gains any meaningful information from this parameter.

Lastly, we intentionally disregarded the meaning in poems and significantly simplified our measures around strophe uniqueness. As demonstrated in Figure 11, generated verses tend to repeat entire words and syllables to create the illusion of rhyming, whereas a more preferable approach would involve generating syllables with close phonetics.

References

Jonas Belouadi and Steffen Eger. 2023. Bygpt5: End-to-end style-conditioned poetry generation with token-free language models.

Lukáš Chaloupský. 2022. Automatic generation of medical reports from chest x-rays in czech.

Wietse de Vries and Malvina Nissim. 2021. As good as new. how to successfully recycle English GPT-2 to make models for other languages. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 836–846, Online. Association for Computational Linguistics.

Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, Zihao Wu, Lin Zhao, Dajiang Zhu, Xiang Li, Ning Qiang, Dingang Shen, Tianming Liu, and Bao Ge. 2023. Summary of ChatGPT-related research and perspective towards the future of large language models. *Meta-Radiology*, 1(2):100017.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Kai-Ling Lo, Rami Ariss, and Philipp Kurz. 2022. Gpoet-2: A gpt-2 based poem generator.

Dominik Macháček. 2014. Sekacek. `https://github.com/Gldkslfmsd/sekacek`.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive learning from complex explanation traces of gpt-4.

Arturo Oncevay and Kervy Rivas Rojas. 2020. Revisiting neural language modelling with syllables.

OpenAI. 2023. Gpt-4 technical report.

Petr Plecháč. 2018. *A Collocation-Driven Method of Discovering Rhymes (in Czech, English, and French Poetry)*, pages 79–95. Springer International Publishing, Cham.

Petr Plecháč. 2016. Czech verse processing system kvĚta – phonetic and metrical components. *Glottotheory*, 7(2):159–174.

Petr Plecháč and Robert Kolár. 2015. The corpus of czech verse. *Studia Metrica et Poetica*, 2(1):107–118.

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

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Georgios Spithourakis and Sebastian Riedel. 2018. Numeracy for language models: Evaluating and improving their ability to predict numbers. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.

Milan Straka, Jakub Ná plava, Jana Straková, and David Samuel. 2021. RobeCzech: Czech RoBERTa, a monolingual contextualized language representation model. In *Text, Speech, and Dialogue*, pages 197–209. Springer International Publishing.

Pedro Javier Ortiz Suárez, Laurent Romary, and Benoît Sagot. 2020. A monolingual approach to contextualized word embeddings for mid-resource languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Changhan Wang, Kyunghyun Cho, and Jiatao Gu. 2019. Neural machine translation with byte-level subwords.

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.

A Generated Strophes

We wanted to showcase some of the generated strophes with stress annotation.

SYLLABLE tokenizer Model

Forced Generation!
AXAX # 1900
J # 11 # ní # ó, jaká radost! jaké potěšení!
J # 10 # ší # jaký to jásot v duši nejvyšší,
J # 11 # ní # a každé slovo jako požehnání
J # 8 # tí # se v srdci lidském zachytí.

ó, ja-ká ra-dost! ja-ké po-tě-še-ní! ◡ - ◡ - ◡ - ◡ - ◡ ◡ ◡	**A**	*iamb*
ja-ký to já-sot v du-ši nej-vyš-ší, - ◡ - - ◡ - ◡ - ◡ ◡	**X**	*iamb*
a kaž-dé slo-vo ja-ko po-že-hná-ní ◡ - ◡ - ◡ - ◡ - ◡ ◡ ◡	**A**	*iamb*
se v srd-ci lid-ském za-chy-tí. ◡ - ◡ - ◡ - ◡ ◡	**X**	*iamb*

Figure 10: Strophe with non-rhyming verses.

In Figure 10 we can see, that the model tries to fulling the meter iamb (J) by utilizing prepositions *ó, a, se*, which don't carry stress. Stress is then shifted to even syllables.

Figure 11 is an example of the model trying to generate meter trochee (T). To achieve this model disregards prepositions as verse starter, which would shift the stress one syllable back. Also to allow single syllable word *cti*, it's followed by preposition to properly align stress. But the resulting strophe seems off, as the repetition in first verse is too much.

OUR tokenizer Model

Forced Generation!
AABB # 1840
T # 8 # sti # v každé ctnosti, v ctnosti ctnosti
T # 8 # sti # dojdeš cti a chvály dosti,
T # 7 # můj # že jsy pravý přítel můj
T # 7 # můj # pane bože, otče můj!

v kaž-dé ctnos-ti, v ctnos-ti ctnos-ti	**A**	*trochee*
- ⌣ - ⌣ - ⌣ - ⌣		
doj-deš cti a chvá-ly dos-ti,	**A**	*trochee*
- ⌣ - ⌣ - ⌣ - ⌣		
že jsy pra-vý pří-tel můj	**B**	*trochee*
⌣ ⌣ - ⌣ - ⌣ ⌣		
pa-ne bo-že, ot-če můj!	**B**	*trochee*
- ⌣ - ⌣ - ⌣ ⌣		

Figure 11: Example of trochee

OUR tokenizer Model

Basic Generation!
ABAXBX # 1880
J # 11 # ní # a v chvíli té jsem cítil, jak se sklání
J # 10 # hlas # má duše k tobě, a jak slyším hlas,
J # 9 # ni # jenž volá mne pojď, pojď ke mně, zvedni!
J # 6 # zas # já zachvěl se a zas
J # 8 # třás # jsem rozechvěn se, chvěl a třás
J # 2 # slyš # ó slyš!

a v chví-li té jsem cí-til, jak se sklá-ní	**A**	*iamb*
⌣ - ⌣ ⌣ ⌣ - ⌣ ⌣ ⌣ - ⌣		
má du-še k to-bě, a jak sly-ším hlas,	**B**	*iamb*
⌣ - ⌣ - ⌣ ⌣ ⌣ - ⌣ -		
jenž vo-lá mne pojď, pojď ke mně, zve-dni!	**A**	*iamb*
⌣ - ⌣ ⌣ - - ⌣ ⌣ - ⌣		
já za-chvěl se a zas	**X**	*iamb*
⌣ - ⌣ ⌣ ⌣ -		
jsem ro-ze-chvěn se, chvěl a třás	**B**	*iamb*
⌣ - ⌣ ⌣ ⌣ - ⌣ -		
ó slyš!	**X**	*iamb*
⌣ -		

Figure 12: Example of strophe of six verses

Figure 12 shows example of Basic generation. The combination of longer strophe and non-rhyming verse proves a little difficult for the model. The meter iamb (J) is achieved by using conjunctions, prepositions and single syllable words.