Assignment-1 Bad Poets Society

Tingyan Lu s4226992
Zhiwei Ma s4437586
Yuanyuan Rong s4508580

1 Introduction

Natural language processing (NLP) has made significant strides in tasks like text generation, dialogue systems, and translation. However, poetry generation remains uniquely challenging due to the need to incorporate rhythm, rhyme, and emotion. For this assignment, we chose the GPT-2 model for its strong performance in text generation and contextual understanding. Our poetry generator uses the pre-trained GPT-2 model to create poems from user-provided prompts. This model, trained on large-scale text data, has extensive language knowledge. Each poem is formatted for readability, with matching images. Key steps in our process included:

1. Gathering a collection of landscape-themed poems from sources like Poetry.net and PoemHunter.com.
2. Identifying rhythmic patterns in these poems.
3. Developing training methods for poetry generation.
4. Setting criteria for when to stop poem generation.
5. Create an image that reflects the tone and emotion of each poem.

We focused on poems with natural themes—trees, sunsets, and the sky—using ordinary scenes to express deeper meanings. Key verbs and nouns were highlighted to guide coherent poem generation.

2 Implementation

First, we need to install the necessary libraries, GPT2Tokenizer, GPT2LMHeadModel for generating poetry and StableDiffusionPipeline for generating images. In this model, the tokenizer splitting the input text into tokens, encoding these tokens into numeric IDs for model processing, and then decoding the output IDs back into natural language text.

```
# Load pre-trained model and tokenizer
model_name = 'gpt2'
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
model = GPT2LMHeadModel.from_pretrained(model_name)
```

After loading the pre-trained GPT-2 model, we set it to evaluation mode. This step is crucial because it ensures that the model's behavior remains stable during text generation. In evaluation mode, certain layers of the model, are disabled, allowing the model to produce more consistent and reproducible results. This is particularly important for generating poetry, where maintaining coherence and stylistic consistency across lines is essential.

```
#Set the model to evaluation mode
model.eval()
```

Then, we define the function `get_rhyming_words(word)` to find words that rhyme with the given input word by querying the RhymeBrain API which is a web-based service that provides rhyming words for a given input word. The function sends a GET request to the API, which returns a list of potential rhymes along with their respective scores. In this implementation, we filter the results to include only words with a score greater than 300, as these higher scores indicate stronger rhyming matches. This filtering is done using a list comprehension, which extracts only the words that meet the score condition.

```python
# Function to get rhyming words
def get_rhyming_words(word):
    # Use the RhymeBrain API to get rhyming words
    response = requests.get(f"https://rhymebrain.com/talk?function=getRhymes&word={word}")
    if response.status_code == 200:
        rhymes = response.json()
        # Filter out low-score rhymes, keeping only words with a score above 300
        return [rhyme['word'] for rhyme in rhymes if rhyme['score'] > 300]
    return []
```

We also define the function `format_poem` to ensure that the generated poem text does not exceed the specified character limit per line (40 characters by default), thereby improving the readability and aesthetics of the poem. The function processes each word individually to avoid word truncation and ensure the integrity of the poem's structure. If the `enforce_rhyme` parameter is true, the function checks the last word of the second-to-last line and replaces the last word with a rhyming word to enhance the rhythm and auditory beauty of the poem. This adjustment effectively improves the artistic expression of the poem.

```
    def format_poem(text, line_length=40, enforce_rhyme=False)
```

With the model and necessary functions in place, we now proceed to the core task: generating poetry `generate_poem(prompt, max_length, num_lines)`. To construct a poem, we follow a sequence of steps where we use several parameters to control the generation process. Here's how the process is carried out:

- input text: A starting phrase or prompt that we provide to the model to guide the poetry generation.
- max_length: This parameter sets the maximum number of tokens (words, punctuation, etc.) that the generated text can contain, allowing us to control the length of the poem.
- num_return_sequences: We set this parameter to 1, so the model generates one poem at a time. This helps us focus on refining a single output per generation attempt.
- no_repeat_ngram_size: It prevents the repetition of phrases by ensuring that no n-grams (sequences of n words) of size 2 are repeated in the output. This keeps the generated poem more varied and avoids monotonous phrases.
- early_stopping: Setting this to True allows the model to stop generating text if it reaches a natural end, such as a sentence completion. This ensures the output feels complete and not abruptly cut off.
- top_k, top_p, temperature: These parameters control the randomness and diversity of the generated text.

3 Problems and Solutions

During the poetry generation process, we encountered several issues that affected the quality and coherence of the poems. To address these challenges, we implemented a series of solutions, as detailed below:

1. The generated poems often resulted in incomplete sentences. For example:

> [In the heart of the yellow wood, two roadsdiverged, each path whispering secrets of choicesuntold Poem,
> The first road was a narrow one, asmall one.The second road, the narrowest, was theone that led to the other road,
> It was narrow, but itwas not narrow.They are]

Figure 1: example of poem generating problem:incomplete sentence

Solution: We use the maximum output length and the number of output lines to limit the poem generation, ensuring that it forms complete sentences.Additionally, we enabled early stopping, which allows the model to end the generation once it detects a natural completion in the sentence.

2. Some generated texts did not resemble poetry; they appeared more like simple narratives without rhythm or rhyme.

> [In the heart of the yellow wood, two roadsdiverged, each path whispering secrets of choicesuntold Poem,
> The first road was a narrow one, asmall one.The second road, ...]

Figure 2: example of poem generating problem:narratives and ellipses

Solution: To address this, we used the `get_rhyming_words(word)` function to retrieve rhyming words for a given word using the RhymeBrain API.

3. There are instances where the generated poems contain ellipses (...) showed in Figure 2.
   Solution:To correct this, we enhanced the `format_poem` function to ensure that each generated line concludes properly. This involves checking for punctuation marks, such as periods, commas, or exclamation points, at the end of each line.

By applying these solutions, we were able to make the generated poems much better. The final poems are clearer, follow a consistent style, and have the rhythm and structure.

4 Our poems

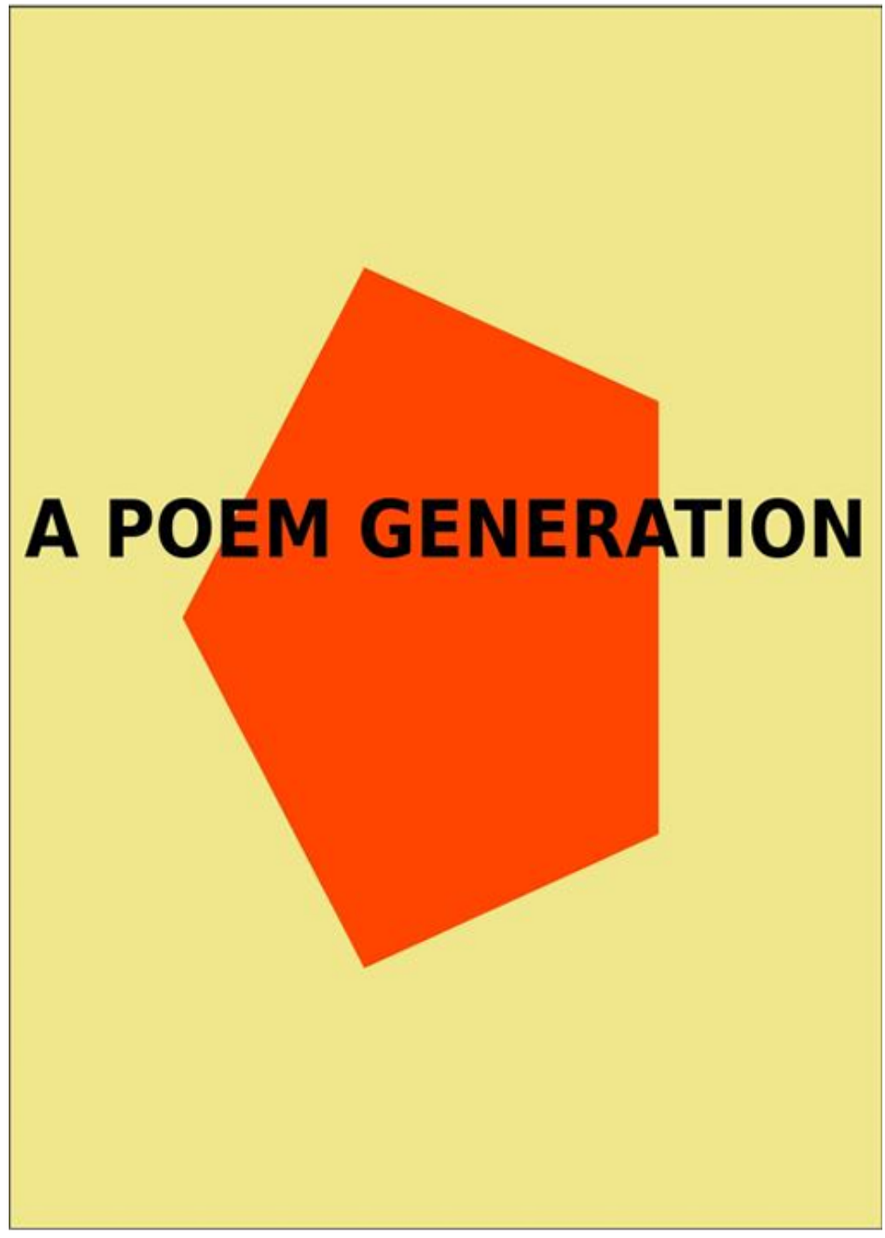


Figure 3: cover of our generation

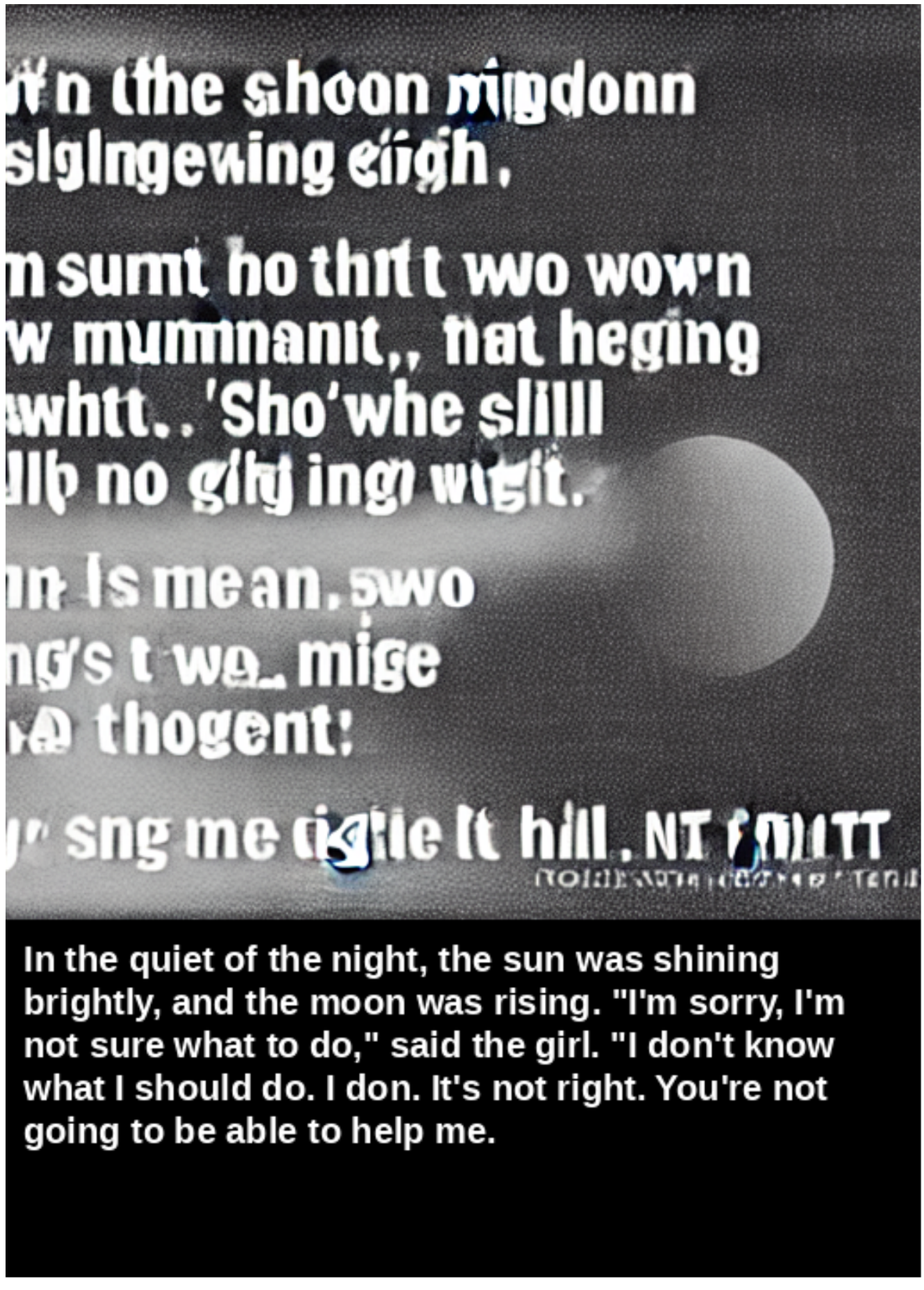
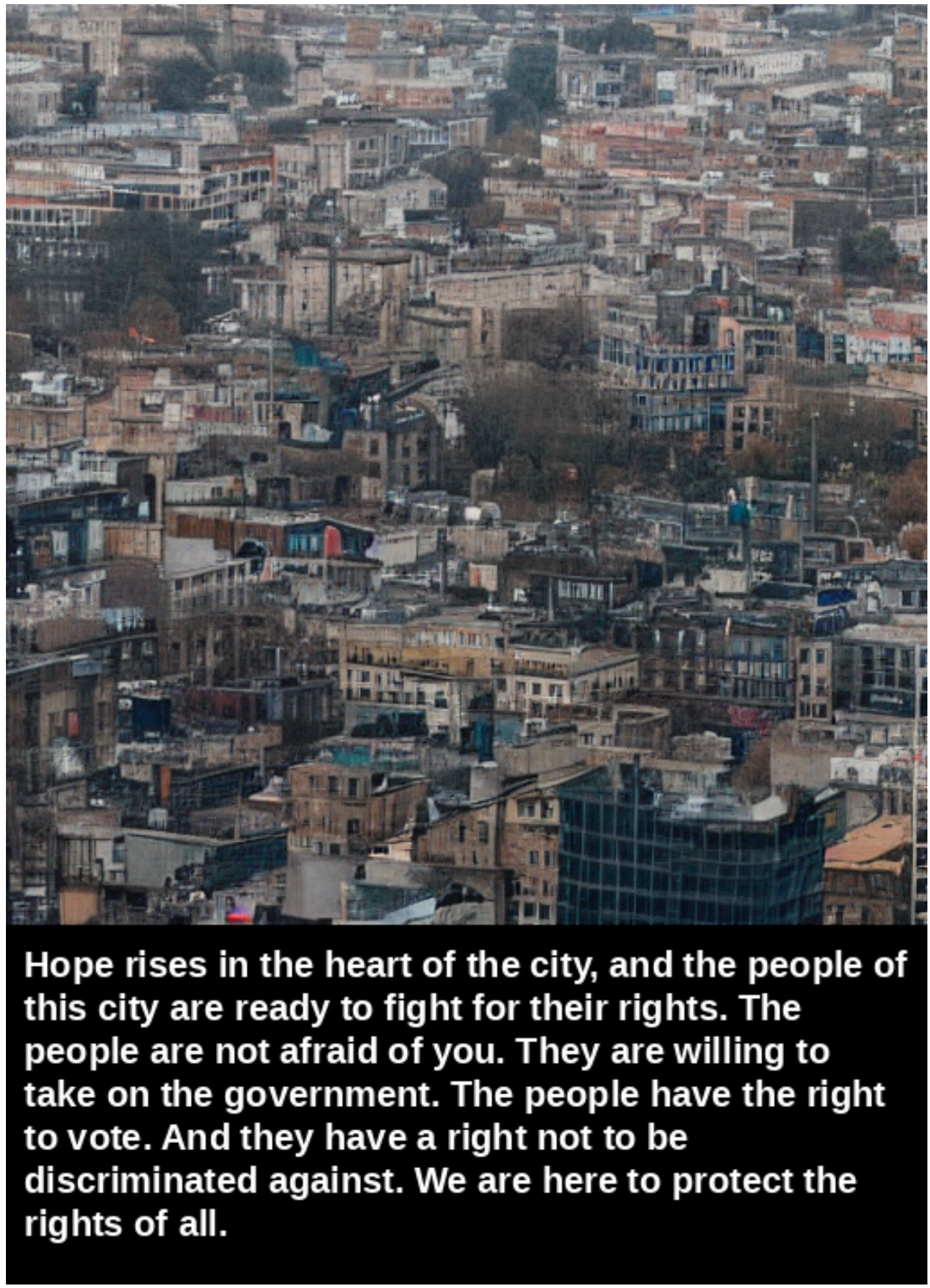


Figure 4: cover of poem 1

Figure 5: cover of poem 2



]

Figure 6: cover of poem 3

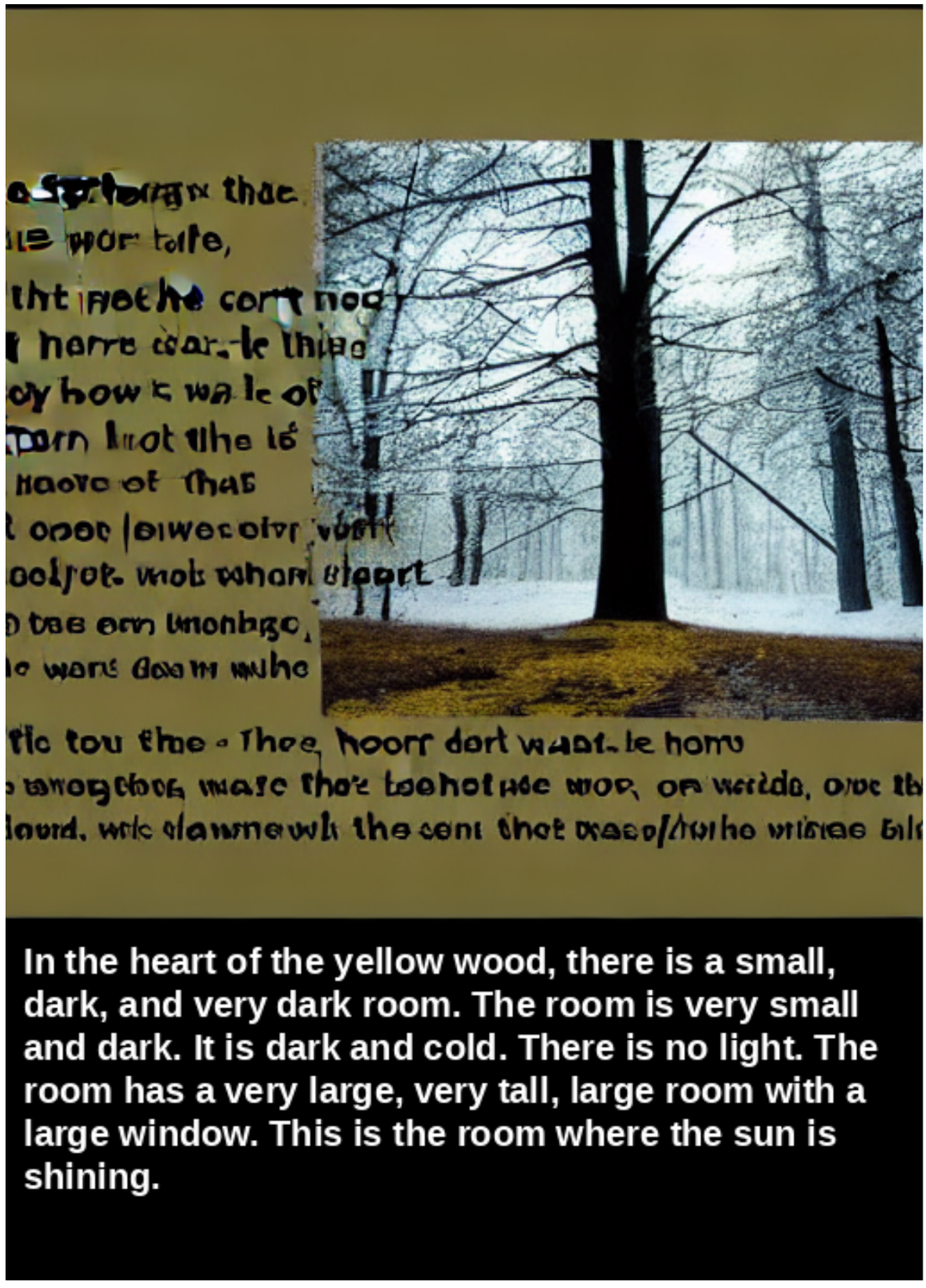


Figure 7: cover of poem 4

Figure 8: cover of poem 5

5 Discussion

We chose the themes because they represent different artistic conceptions in different poems. It may be majestic and shocking, or it may be a long and distant stream under a small bridge, or it may be positive and full of sunshine, or it may be negative and pessimistic. These landscapes can appear in poems of any artistic conception, but the meanings expressed vary greatly. We use ordinary scenes that can be seen everywhere in life in poems to express deeper meanings, so we choose poetry collections that include landscapes and try to see how they will perform after learning.

To some extent, our generated poems reflect the input theme with complete, grammatically correct sentences. The model demonstrates creativity by producing artistic fragments, but the rhythm could be improved. We plan to incorporate more rhyming elements in future iterations to enhance the poetic flow. Expanding the training dataset with more diverse poetry could also enrich the vocabulary and deepen the model's ability to capture a range of artistic expressions. Additionally, learning from varied scenarios would improve its understanding of linguistic diversity.

While our methods rely on learning and imitation, true creativity in poetry generation still requires more than simply following predefined rules. The generated poems often lack the unique stylistic expression found in human creativity. Although the model has been trained on human language patterns, it lacks the deeper, nuanced understanding necessary for crafting poetry with a distinctive voice. As human language evolves, machine learning must continue to adapt. Exploring new strategies to enhance the model's capacity for creative language generation will be crucial for future advancements.