DSA4213 Assignment 2: Build and Compare Small Language Models

Nguyen Cao Duy (A0258078R)

Table of contents

1. Data

1.1 Corpus Introduction

For this assignment, I use Arthur Conan Doyle's entire Sherlock Holmes works as a corpus of around 650,000 words (almost 4 millions characters). The text is then split into training (80%), validation (10%), and test (10%) sets.

1.2 Tokenization

We explore three tokenization strategies for our ablation study (to be discussed later):

- **Word-level**: Words are separated by whitespace, resulting in a vocabulary size of **~40,000**.
- **Subword-level**: Using a Byte Pair Encoding (BPE) variant through the `sentencepiece` library, we train a subword tokenizer with a vocabulary size of **2,000**.
- **Character-level**: Each character is treated as a token, resulting in a vocabulary size of **~100**.

```
--- Sample text ---
Is Holmes alive? I didn't see him.
--- Word-level Tokenization ---
Tokens: ['Is', 'Holmes', 'alive?', 'I', "didn't", 'see', 'him.']
--- Subword-level Tokenization ---
Tokens: ['▁I', 's', '▁Holmes', '▁al', 'ive', '?', '▁I', '▁did', 'n', "'", 't', '▁see', '▁him', '.']
--- Character-level Tokenization ---
Tokens: ['I', 's', ' ', 'H', 'o', 'l', 'm', 'e', 's', ' ', 'a', 'l', 'i', 'v', 'e', '?', ' ', 'I', '
', 'd', 'i', 'd', 'n', "'", 't', ' ', 's', 'e', 'e', ' ', 'h', 'i', 'm', '.']
```

2. Models

We explore two models for this assignment: Long Short-Term Memory (LSTM) and Transformer (decoder-only). The brief descriptions of the models are as follows:

1. Embedding layer: Maps input tokens to vectors (sinusoidal positional encoding is added for the Transformer).

2. Main layer: Stacks of either LSTM layers or Transformer decoder layers.
3. Output layer: A linear layer followed by a softmax to predict the next token.

Dropout and Layer Normalization are used to improve generalization and training stability. Pytorch is used to implement both models, and training is done on a single P100 GPU on Kaggle.

3. Experiments and Results

Each experiment consists of training the models for **20,000** iterations (each iteration processes one batch of data). This is better for comparison, as the number of training iterations per epoch can vary based on the tokenization strategy (i.e., 1 epoch for word-level tokenization has fewer iterations than character-level tokenization due to the longer sequences).

The best model is saved based on the lowest validation loss during training (evaluated every **2,000** iterations). Finally, it is evaluated on the test set to report the final test loss and perplexity.

To keep the training consistent, we will use these shared settings across all experiments: Dropout: 0.2, Batch size: 64, Optimizer: AdamW with learning rate 5×10^{-4}

3.1 Tokenization Experiments

We compare the performance of the two models (LSTM and Transformer) across the three tokenization strategies (word-level, subword-level, and character-level). For both models, we fix some hyperparameters to ensure a fair comparison: Embedding size: 64, Hidden size: 256, Number of layers: 2, Sequence length: 64. Even with these shared settings, the number of parameters might differ between models due to their different architectures.

Model	# Params (M)	Test Loss	Test Perplexity	Training Time
LSTM + Word	13.62	6.93	1025.40	27m 03s
LSTM + Subword	1.50	3.94	51.44	7m 21s
LSTM + Char	0.89	1.18	3.26	10m 48s
Transformer + Word	5.23	6.84	936.35	12m 50s
Transformer + Subword	0.36	4.17	64.92	3m 31s
Transformer + Char	0.11	1.63	5.10	5m 16s

As we move from character-level to word-level tokenization, the model size increases significantly because of the larger vocabulary size. Surprisingly, the subword-level tokenization has lower training time than character-level tokenization despite having a larger model size, maybe due to the `sentencepiece` library's tokenization efficiency.

Test perplexity decreases as we move from word-level to character-level tokenization, indicating that models are more confident in their predictions. However, this might be due to the smaller vocabulary size, and does not necessarily mean better text generation quality. As we can see from the sample generations below, character-level models usually produce typos and nonsensical words, while word-level and subword-level models generate more coherent text.

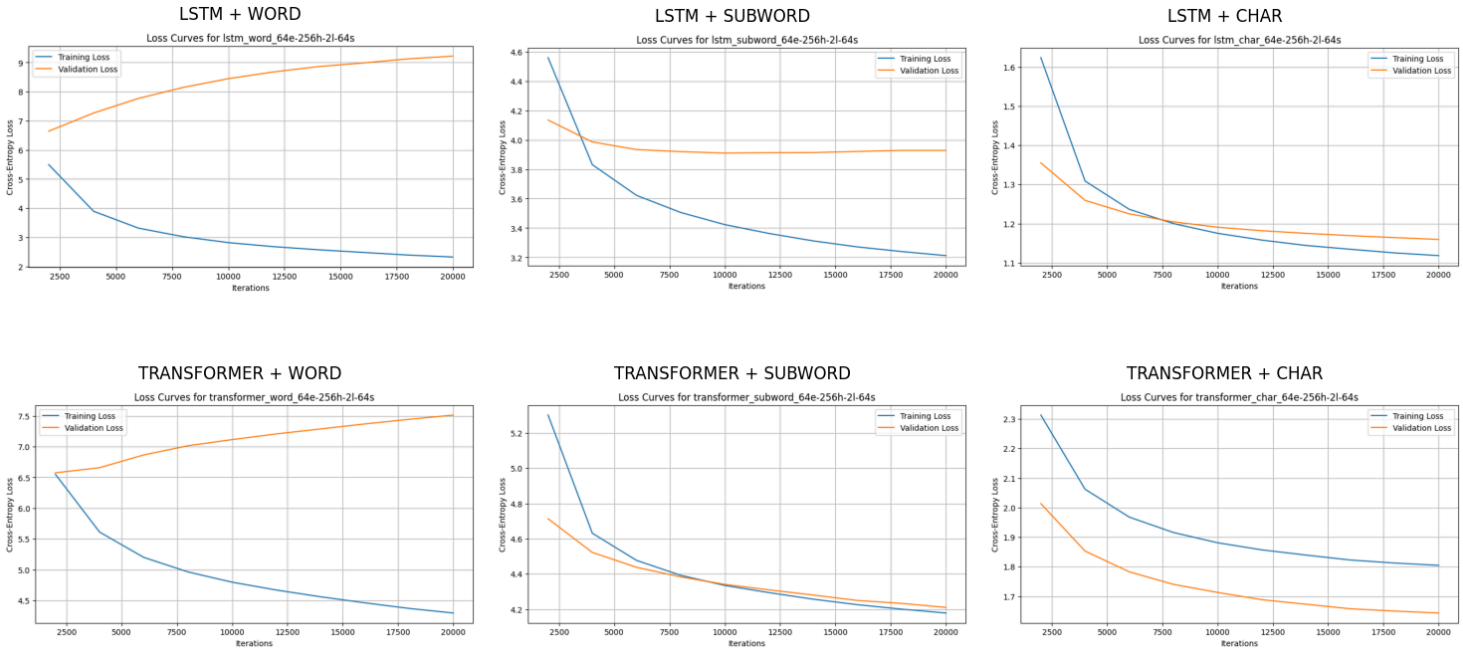
Note: A limitation of our word-level tokenization is that it does not handle punctuation properly since we only split by whitespace (some punctuation marks are attached to words). This might explain the poor performance of word-level models in perplexity.

```
--- Prompt: 'Sherlock Holmes took his bottle from the corner of the room' ---
--- Generation from lstm_word_64e-256h-2l-64s (temp=1.0) ---
on the moor. "You can surely say that. The bird was the man who wrote in the interest I should

--- Generation from lstm_subword_64e-256h-2l-64s (temp=1.0) ---
where the hinges was seated by and lit the termmer all miners. The man was waiting at the back, but
was the portrer Tener which hung

--- Generation from lstm_char_64e-256h-2l-64s (temp=1.0) ---
```

```
eant we were allowing together.  I
     saw the groan who had roined, kindly oaken as of our prompt.

--- Generation from transformer_word_64e-256h-2l-64s (temp=1.0) ---
knocked I have done of San our mantelpiece. and absolutely Edwards," of one of the scrub, of the
bell, that

--- Generation from transformer_subword_64e-256h-2l-64s (temp=1.0) ---
. It was a dark head. The band-ag, when this lady began to having held out into a good side. There
was no flower and seally unless streed case

--- Generation from transformer_char_64e-256h-2l-64s (temp=1.0) ---
 eenter to by of my, Mor
     man, and in a man the dare?"

     "I was a rencee, lasker in the woul
```

Looking at the training curves below, we can see that word-level models overfit very quickly, while character-level models take much longer to converge. Our subword-level models strike a good balance between the two.

At this small scale, LSTM models perform better than Transformer models in terms of perplexity, but they take longer to train and seem to overfit faster.



3.2 Model Scaling Experiments

We compare the performance of the two models (LSTM and Transformer) using subword-level tokenization, while varying the model sizes. Using the following settings as the baseline: Embedding size: 64, Hidden size: 256, Number of layers: 2, Sequence length: 64. We then scale the models up and down by adjusting one hyperparameter at a time, while keeping the others fixed.

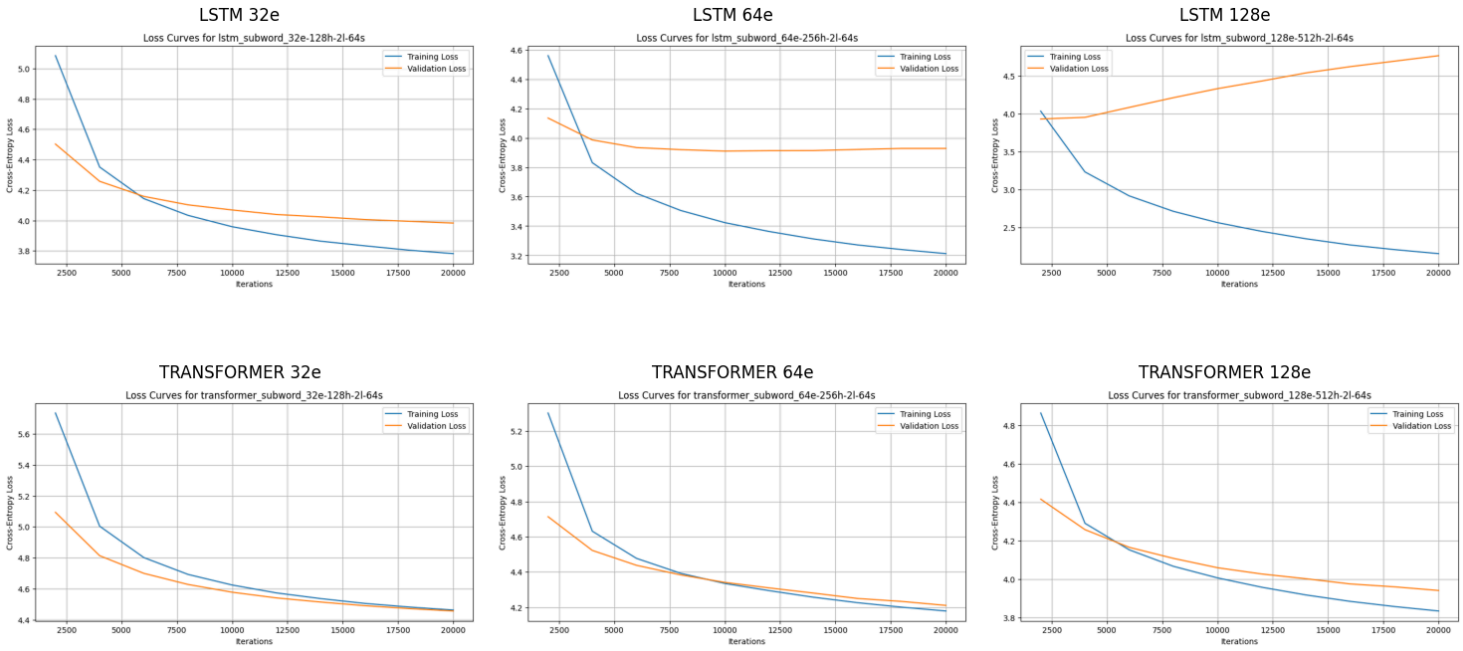
3.2.1 Varying Embedding Size

We conduct experiments by varying the embedding size for both LSTM and Transformer models in {32, 64, 128}, and also update the hidden size as 4 times the embedding size to maintain a balanced architecture.

Model	Params (M)	Test Loss	Test Perplexity	Training Time (s)
LSTM 32e	0.54	3.98	53.38	4m 16s

Model	Params (M)	Test Loss	Test Perplexity	Training Time (s)
LSTM 64e	1.50	3.94	51.44	7m 21s
LSTM 128e	4.70	3.93	50.77	33m 51s
Transformer 32e	0.16	4.43	83.65	3m 40s
Transformer 64e	0.36	4.17	64.92	3m 31s
Transformer 128e	0.91	3.94	51.25	9m 38s

As we can see from the test perplexity above, increasing the embedding size improves the model's performance, especially for the Transformer model. However, the improvement is marginal in the case of the LSTM model. The training curves below also show that LSTM models are more susceptible to overfitting with larger embedding sizes, while Transformer models benefit more from the increased capacity.

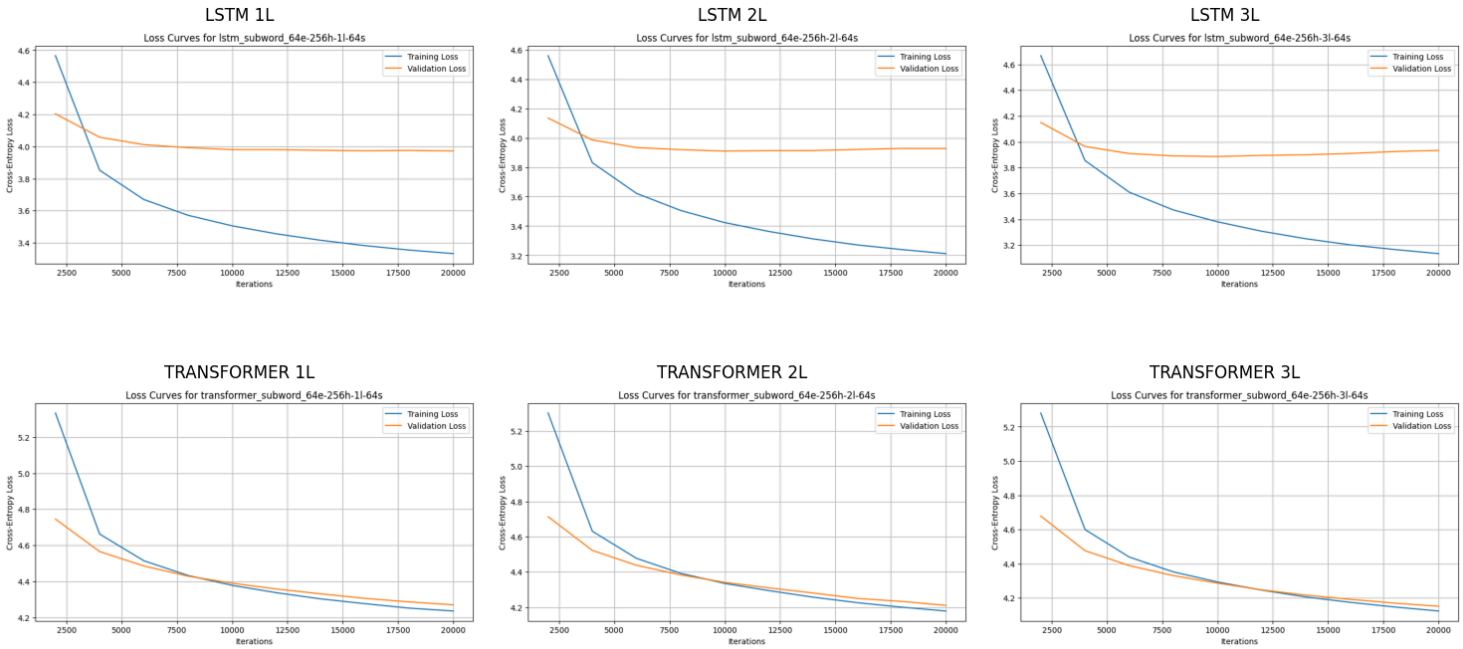


3.2.2 Varying Number of Layers

We conduct experiments by varying the number of layers for both LSTM and Transformer models in {1, 2, 3}.

Model	Params (M)	Test Loss	Test Perplexity	Training Time (s)
LSTM 1L	0.97	3.98	53.56	4m 45s
LSTM 2L	1.50	3.94	51.44	7m 21s
LSTM 3L	2.02	3.92	50.41	9m 55s
Transformer 1L	0.31	4.24	69.23	2m 51s
Transformer 2L	0.36	4.17	64.92	3m 31s
Transformer 3L	0.41	4.12	61.30	4m 24s

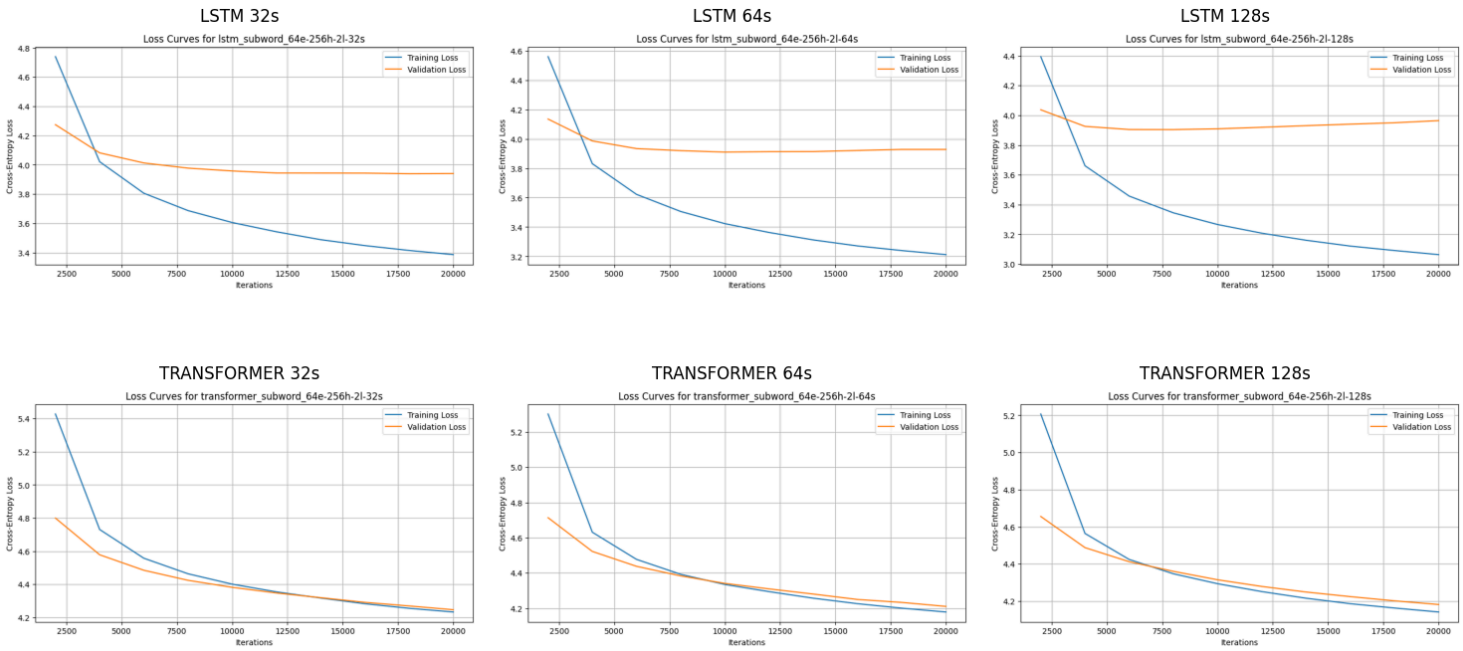
As we can see from the test perplexity above, increasing the number of layers improves the model's performance for both LSTM and Transformer models, but the improvement is small. The training curves below also show that between 1 and 3 layers, the models do not overfit significantly, indicating that they can benefit from deeper architectures without a substantial risk of overfitting at this scale.



3.2.3 Varying Sequence Length

We conduct experiments by varying the sequence length for both LSTM and Transformer models in {32, 64, 128}. Note that varying the sequence length does not change the number of parameters, it only affects the training time.

Model	Params (M)	Test Loss	Test Perplexity	Training Time (s)
LSTM 32s	1.50	3.94	51.63	4m 17s
LSTM 64s	1.50	3.94	51.44	7m 21s
LSTM 128s	1.50	3.91	49.65	14m 2s
Transformer 32s	0.36	4.21	67.35	2m 57s
Transformer 64s	0.36	4.17	64.92	3m 31s
Transformer 128s	0.36	4.16	63.16	5m 38s

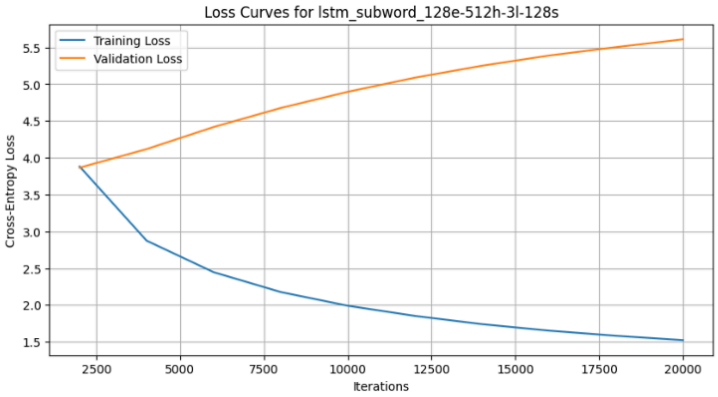


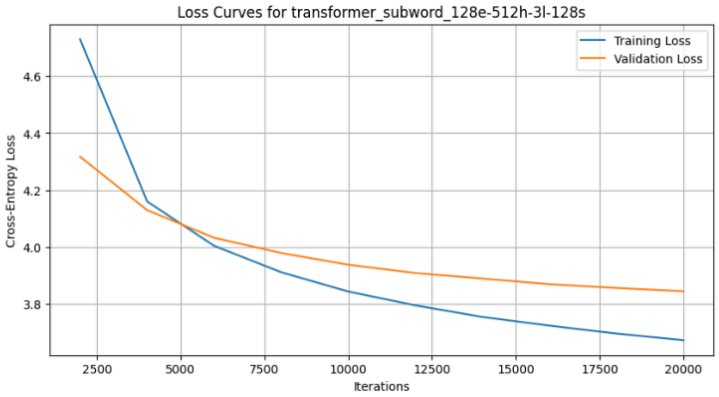
Similar to the case of number of layers, increasing the sequence length improves the model's performance slightly and does not make models more susceptible to overfitting. Not surprisingly, training time increases with longer sequences, especially for the LSTM model since it processes sequences sequentially.

3.2.4 Varying All 3 Factors Together

We choose the largest configuration from each of the previous experiments: Embedding size: 128, Hidden size: 512, Number of layers: 3, Sequence length: 128 to see how the models perform at a larger scale.

Model	Params (M)	Test Loss	Test Perplexity	Training Time (s)
LSTM 128e 3L 128s	6.80	3.88	48.52	33m 51s
Transformer 128e 3L 128s	**1.11**	**3.83**	**46.26**	**9m 38s**





Not surprisingly, the Transformer model scales better than the LSTM model, achieving lower perplexity with fewer parameters and significantly less training time. The training curves also suggest that the Transformer model trains more stably and we might still be able to train it for more iterations, or scale it even bigger to get even better performance.

3.3 Text Generation Experiments

Using the best model from the previous experiments `transformer_subword_128e-512h-3l-128s` with a test perplexity of **46.26**, we generate text samples with different temperature settings to observe the effect on the generated text.

```
--- Prompt: 'Sherlock Holmes picked up a small piece of paper from the ground' ---
--- Generation from transformer_subword_128e-512h-3l-128s (temp=0.7) ---
s, and he was a number of action, and he had a telegram for heir, as he had seen, and his face showed
that no difficulty was keenly well more
--- Generation from transformer_subword_128e-512h-3l-128s (temp=1.0) ---
s. We turned a tree and strength for a plantic and light from his climbler. "'The twenty of the
houses, and there have whined
--- Generation from transformer_subword_128e-512h-3l-128s (temp=1.3) ---
. As he stood the doorway through with a lamp of neat,--one! that all happened," he asked. "Mr. we
please. There does exactly no one either d
```

```
--- Prompt: ''‘My dear Watson,’ said Holmes' ---
--- Generation from transformer_subword_128e-512h-3l-128s (temp=0.7) ---
, but I have anxious to work matter, for I am a sort of a very long time, and I have to attain. I
shall be able to do, and can hardly tell
--- Generation from transformer_subword_128e-512h-3l-128s (temp=1.0) ---
, to grot  ⁇  , for I at last one day called three among this morning in Mr. Sherlock Holmes. He held
a ribbled frank and gloomily dragged
--- Generation from transformer_subword_128e-512h-3l-128s (temp=1.3) ---
. It's cut the convention which won't show you; it may about a mere judgment oneize those which man
must tell me perhaps upon eat about anything, nor
```

Appendix

A.1 Code & Model Weights

All code and model weights can be found at my GitHub repository: https://github.com/ncduy0303/dsa4213-assignment-2. More instructions are provided in the repository's README.

A.2 AI Tool Declaration

I used Gemini 2.5 Pro and Github Copilot to generate ideas, format paragraphs, improve expression, produce drafts, refine, and finalize my assignment. I am responsible for the content and quality of the submitted work.