

HOMEWORK 1^{*}

TOKENIZATION AND FINETUNING[†]

24-880 AI AGENTS FOR ENGINEERS
<https://cmuagents.github.io>

OUT: Thursday, January 22nd, 2026
DUE: Friday, January 30th, 2026
TAs: Peter Pak

Instructions

- **Collaboration Policy:** Please read the collaboration policy in the syllabus.
- **Late Submission Policy:** See the late submission policy in the syllabus.
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code.
 - **Written:** You will submit your completed homework as a PDF to Gradescope. Please use the provided template for the submissions which can be written in $\text{\LaTeX}$ (via [Overleaf](#)) or directly on the PDF document. Each answer should be within the box provided. If you do not follow the template, your assignment may not be graded correctly and there will be a **2% penalty** (e.g., if the homework is out of 100 points, 2 points will be deducted from your final score).
 - **Code:** Gradescope's Autograder will be used to evaluate the coding portions of this assignment. In order to best obtain the points for this assignment, please follow the code upload instructions. If you do not follow the instructions correctly the autograder may not work properly.

Question	Points
Byte Pair Encoding (Written)	23
Byte Pair Encoding (Code)	10
Subword Neural Machine Translation (Written)	32
Subword Neural Machine Translation (Code)	10
BERT and Fine-tuning (Written)	20
Code Upload	0
Collaboration and Generative AI Questions	5
Total:	100

^{*}Adapted from CMU 10-623 Generative AI Homework Template

[†]Compiled on Thursday 22nd January, 2026 at 16:55

Introduction

In this homework assignment we'll look into the concepts of tokenization and finetuning.

Tokenization is a critical component for operating of large language model as it converts natural language text input into a format that neural networks can understand.

At a high-level, this homework will cover the following aspects.

1. Understand and implement Byte Pair Encoding
2. Explore Subword Neural Machine Translation and how it relates to Byte Pair Encoding
3. How tokenization relates to BERT
4. Introduction to fine-tuning

Local Environment

For this homework we'll be using `uv` as the project and package manager. If you haven't install `uv`, you can follow the installation instructions here <https://docs.astral.sh/uv/>. You can explore all of the available commands using `uv --help` but in short, here are a couple of useful commands that you'll often use:

<code>source .venv/bin/activate</code>	(Linux and Mac) Activates python environment.
<code>source .venv/Scripts/Activate</code>	(Windows) Activates python environment.
<code>uv sync</code>	Installs / synchronizes projects dependencies.
<code>uv add <package></code>	Installs <code>pip</code> package and adds it to project dependencies.
<code>uv remove <package></code>	Removes <code>pip</code> package dependency from project.
<code>uv run <script.py></code>	Runs python script inside of <code>uv</code> environment.

1. Navigate to folder project directory and synchronize dependencies with:

```
uv sync
```

2. Activate the environment

Mac and Linux:

```
source .venv/bin/activate
```

Windows:

```
source .venv/Scripts/activate
```

3. Test `uv` works and run `main.py` file.

```
uv run main.py
```

1 Byte Pair Encoding (Written) (23 points)

Byte Pair Encoding (BPE) was first introduced by Gage [1994] as a method of data compression useful in memory constrained environments due to its fast expansion routine. The compression routine of the algorithm looks for most adjacent byte pairs that occur most frequently within a given pass and replaces the pair with a byte that doesn't already exist within the data. This repeats until there is either no more frequent byte pairs or there are no more remaining unused bytes. The expansion routine is performed over a single pass over the input file, where byte literals are passed directly to the output buffer and byte pairs are pushed onto a stack. Within each iteration, if the stack contains data the byte there is used as the next input byte, otherwise the next input byte is obtained from the input file.

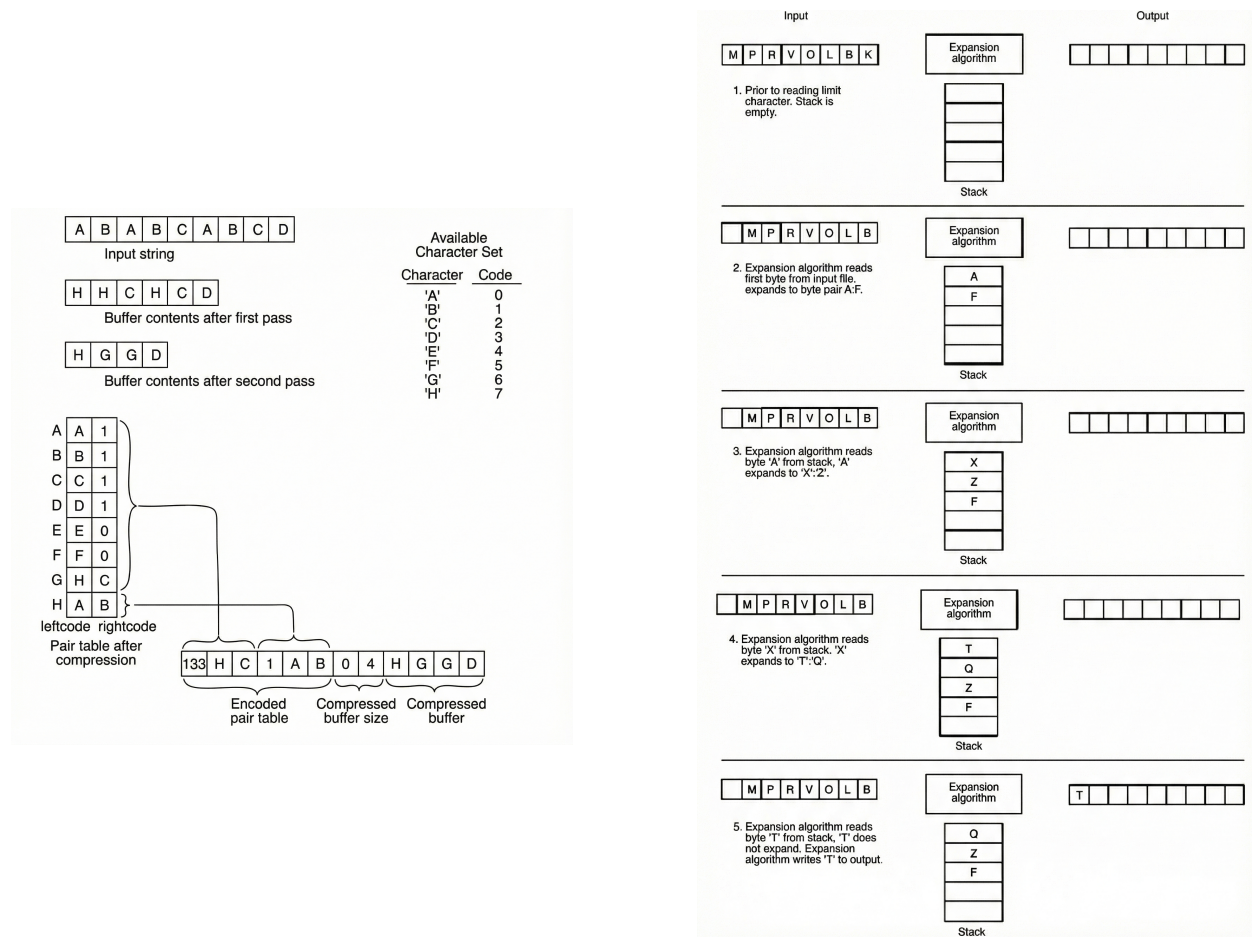


Figure 1: Illustrations from Gage [1994] upscaled with *Nano Banana Pro* outlining the general BPE compression (left) and expansion (right) processes. **(Left):** Byte pair **AB** is replaced with the byte for **H** during the first pass, reducing the buffer length from 9 to 6. During the second pass the the pair **HC** is replaced with the byte for **G** further reducing the buffer length from 6 to 4. The compression table on the bottom left indicates the mapping of each byte where the left and right columns indicate the left and right bytes within the pair. (i.e. row 7 shows that **G** represents the byte pair **HC**). **(Right):** Seperate example shows **K** expanded to byte pair **AF** and placed onto stack. **A** is removed from stack and expanded to byte pair **XZ** and placed back onto stack. **X** is removed from stack and expanded to byte pair **TQ** and placed back onto stack. **T** is removed from stack and placed into output as it is not associated with a byte pair.

Before we implement Byte Pair Encoding, let's first review some its conceptual aspects.

- 1.1. (5 points) What is a potential limitation of Byte Pair Encoding that Gage [1994]. states and what are the two examples that the author mentions?

- 1.2. (2 points) According to Gage [1994] during compression, how many pass(es) are performed over the input data?

(i.e. **ABABCABCD** input string from Fig. 1)

- ☐ 0
☐ 1
☐ n

- 1.3. (2 points) According to Gage [1994], how many pass(es) are required to expand the buffer data?

(i.e. **HGGD** compressed buffer from Fig. 1)

- ☐ 0
☐ 1
☐ n

- 1.4. (4 points) During the compression routine in Fig. 1 (left), what do respective 1 and 0 values in the `rightcode` column indicate?

- 1.5. Let's explore the BPE compression and expansion process with a simple example. For this we can reference the provided pair table (Table 1).

Table 1: BPE Pair Table

	leftcode	rightcode
A	A	1
B	B	1
C	N	A
D	C	N
E	D	O
F	E	-
⋮	⋮	⋮
N	N	1
O	O	1
⋮	⋮	⋮
-	-	1

- 1.5.a. (5 points) What is the compressed buffer for the input string **NANO NANO BANANA**?

Showing work is not required but helps assign partial credit for incorrect answers

- 1.5.b. (5 points) Using the same table (Table 1), what does the compressed buffer **E BAECC** output?

Showing work is not required but helps assign partial credit for incorrect answers

2 Byte Pair Encoding (Code) (10 points)

- 2.1. (10 points) For this part of the assignment we'll look into filling out a simple implementation of Byte Pair Encoding outlined in the starter code: `./code/bpe.py`

This file primarily consists of two functions:

1. `bpe_compression()` - process outlined in fig. 1 (left)
2. `bpe_extraction()` - process outlined in fig. 1 (right)

Please fill out the remaining code indicated by the `# Fill In` comments within the code.

You can run and test the code with the following commands:

<code>uv run code/bpe.py</code>	Runs script with example case (useful for development)
<code>uv run test.py bpe</code>	Runs test script that will be used in autograder

The provided tests are the exact ones that will run on the autograder (i.e. if your local tests pass, they should also pass on the autograder).

3 Subword Neural Machine Translation (Written) (32 points)

Subword Neural Machine Translation is a preprocessing method which text is segmented into subword units, specifically useful in encoding out-of-vocabulary (OOV) words. The approach proposed by [Sennrich et al. \[2016\]](#) implements an adapted version of Byte Pair Encoding (BPE) to first generate the pair table for frequently occurring character sequences within the train text. This is similar to the pair table seen in the compressed output that original BPE produces, however with slight adjustment of merging characters rather than bytes in order to suit the application of word segmentation. Along with this, the compression routine is set to conclude after a given number of operations rather than the original BPE process of repeating until there are no more remaining bytes in the text. This provides a tunable `num_operations` parameter which balances the frequency for complete words and subwords within the dictionary, improving the coverage of tokens during training. This allows for out-of-vocabulary words to be segmented into combinations of word and subword tokens

For this part of the assignment you'll see an example of subword neural machine translation process by investigating the print statements produced by `./code/snmt_written.py`.

Run `uv run code/snmt_written.py` and this should produce the print statement outputs.

3.1. (5 points) The text provided to obtain the BPE pair table looks something like this:

```
The quick brown fox jumped over the lazy dog.  
I ordered a hamburger from mcdonalds.  
I ordered a cheeseburger from mcdonalds.  
I ordered a double bacon cheeseburger from burger king.  
I ordered a double-double from in-in-out burger.  
How much wood could a woodchuck chuck if a woodchuck could chuck wood.  
Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.
```

Before running the code, what tokens would you expect to be added to the vocabulary?

3.2. (5 points) Alright, let's run it. What are some tokens you found in the vocabulary and does that match your expectation?

3.3. Let's take a look at the `num_operations` parameter, keep an on `codes` (indicated as "Learned codes:" in the print statement)

3.3.a. (2 points) Change the `num_operations` to 50, what do you notice with the vocabulary output?

3.3.b. (2 points) Now change the `num_operations` to 1, what do you notice with the vocabulary output?

3.3.c. (4 points) Why does this happen?

3.3.d. (4 points) Why use a limit such as `num_operations` as opposed to running this like the compression routine in BPE?

3.3.e. (5 points) How did the `codes` change while you were changing the `num_operations`. What does `codes` look similar to what we've seen before in the homework?

3.4. Now let's **set num_operations back to 20** and examine the `vocab_threshold` parameter.

3.4.a. (3 points) Change the `vocab_threshold` to 10, what do you notice with the token count?

3.4.b. (2 points) Why does this happen?

4 Subword Neural Machine Translation (Code) (10 points)

- 4.1. (10 points) For this part of the assignment you'll tune the parameters in the starter code:
`./code/snmt.py`

This file contains three parameters to fill in:

1. `TRAIN_TEXT` - training text to learn BPE codes from
2. `INPUT_TEXT` - input text to segment
3. `NUM_OPERATIONS` - number of BPE merge operations

Your goal is to tune these parameters to satisfy the following criteria:

1. `NUM_OPERATIONS` should be at most 20
2. Vocabulary size should be at most 100
3. 'the' should appear as a complete token
4. 'and' should appear as a complete token
5. 'a' should appear as a complete token
6. `INPUT_TEXT` segmentation should produce less than 50 tokens
7. 'er' should appear in the vocabulary
8. `INPUT_TEXT` should have at least 5 complete tokens and 5 partial tokens
9. 'th' should appear in the vocabulary
10. 'an' should appear in the vocabulary

You can run and test the code with the following commands:

<code>uv run code/snmt.py</code>	Runs script with example case (useful for development)
<code>uv run test.py snmt</code>	Runs test script that will be used in autograder

The provided tests are the exact ones that will run on the autograder (i.e. if your local tests pass, they should also pass on the autograder).

5 BERT and Fine-tuning (Written) (20 points)

- 5.1. (5 points) According to [Devlin et al. \[2019\]](#), what tokenization method does BERT use and how does it relate to the BPE approach we explored earlier in this homework?

- 5.2. (5 points) Why is subword tokenization (like BPE or WordPiece) important for models like BERT when handling rare or unseen words?

- 5.3. (5 points) According to [Zhang et al. \[2025\]](#), what is instruction tuning (also known as supervised fine-tuning) and what type of data format does it use?

- 5.4. (5 points) What is the difference between pre-training and fine-tuning in the context of language models like BERT?

6 Code Upload (0 points)

To upload your code to Gradescope, run the following command to generate a zip file:

```
uv run code/zip_code.py
```

Upload the generated zip file to the appropriate programming slot on Gradescope.

6.1. Make sure the following files are included when uploading to Gradescope

- ☐ `bpe.py`
- ☐ `snmt.py`
- ☐ `snmt_utils.py`

7 Collaboration and Generative AI Questions (5 points)

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found in the syllabus.

- 7.1. (2 points) Did you collaborate with anyone on this assignment? If so, list their name or Andrew ID and which problems you worked together on.

- 7.2. (3 points) Did you use generative AI for parts of this assignment? If so, include the input prompts along with links to the conversation.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, May 2019. URL <http://arxiv.org/abs/1810.04805>. arXiv:1810.04805 [cs].

Philip Gage. A new algorithm for data compression. *The C Users Journal archive*, February 1994. URL <https://www.semanticscholar.org/paper/A-new-algorithm-for-data-compression-Gage/1aa9c0045f1fe8c79cce03c7c14ef4b4643a21f8>.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL <https://aclanthology.org/P16-1162/>.

Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, and Guoyin Wang. Instruction Tuning for Large Language Models: A Survey, October 2025. URL <http://arxiv.org/abs/2308.10792>. arXiv:2308.10792 [cs].