Tokenization, Autocompletion and Language Modelling

Suyash Pande - 2025201063
Introduction to NLP – Spring 2026

1 Introduction

Tokenization is a foundational step in Natural Language Processing (NLP), directly influencing vocabulary size, data sparsity, and the effectiveness of downstream language models. In this part of the assignment, we perform corpus cleaning, dataset partitioning, and tokenizer construction for both English and Mongolian CC-100 corpora. Three tokenization strategies are implemented from scratch: whitespace-based, regex-based, and Byte Pair Encoding (BPE).

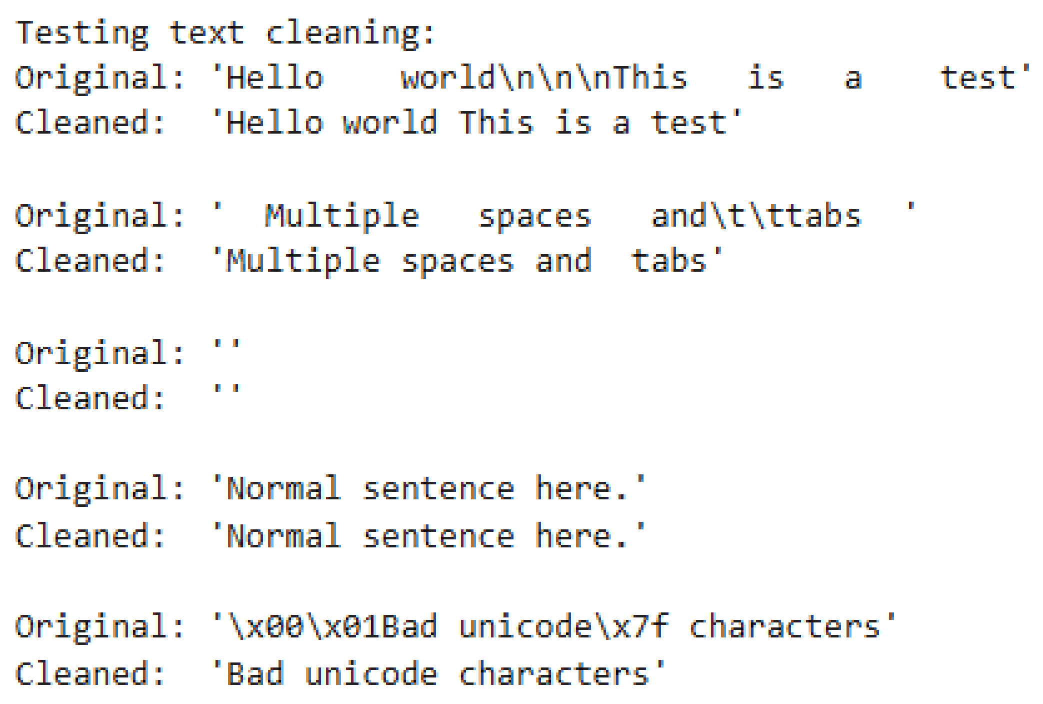
All preprocessing and tokenization decisions are motivated from the perspective of *language modelling suitability* rather than linguistic elegance alone.

2 Corpus Cleaning

The raw CC-100 corpora contain several forms of noise that can negatively affect tokenization and n-gram statistics. The following cleaning steps were applied:

- Removal of empty lines
- Normalization of excessive whitespace
- Removal of tab characters and alignment artefacts
- Filtering of malformed and non-decodable Unicode characters

Figure 1 shows representative examples of raw sentences before and after cleaning.



```
Testing text cleaning:
Original: 'Hello    world\n\n\nThis   is   a    test'
Cleaned:  'Hello world This is a test'

Original: '  Multiple   spaces   and\t\ttabs  '
Cleaned:  'Multiple spaces and  tabs'

Original: ''
Cleaned:  ''

Original: 'Normal sentence here.'
Cleaned:  'Normal sentence here.'

Original: '\x00\x01Bad unicode\x7f characters'
Cleaned:  'Bad unicode characters'
```

Figure 1: Examples of text cleaning showing whitespace normalization and Unicode artefact removal.

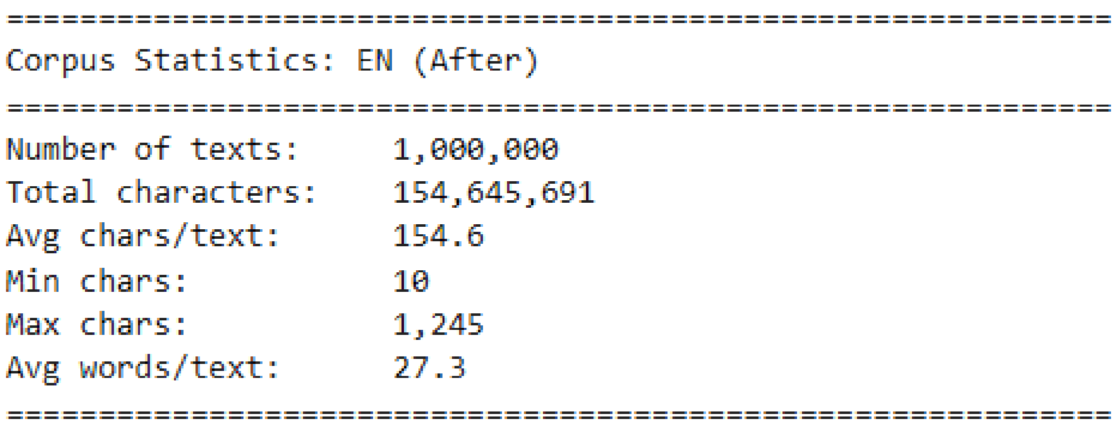
3 Corpus Statistics

After cleaning, corpus statistics were computed to understand the scale and structural properties of the data.

3.1 English Corpus

The cleaned English corpus contains 1,000,000 sentences. Summary statistics are shown in Figure 2.

- Total characters: 154,645,691
- Average characters per sentence: 154.6
- Average words per sentence: 27.3



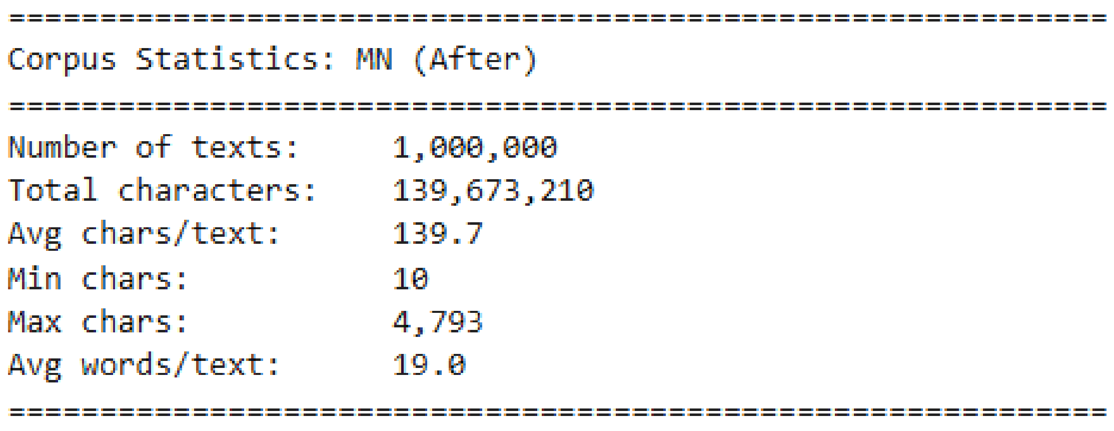
```
============================================================
Corpus Statistics: EN (After)
============================================================
Number of texts:     1,000,000
Total characters:    154,645,691
Avg chars/text:      154.6
Min chars:           10
Max chars:           1,245
Avg words/text:      27.3
============================================================
```

Figure 2: Post-cleaning corpus statistics for English.

3.2 Mongolian Corpus

The Mongolian corpus also consists of 1,000,000 sentences, with noticeably different structural properties due to its agglutinative morphology.

- Total characters: 139,673,210
- Average characters per sentence: 139.7
- Average words per sentence: 19.0



```
============================================================
Corpus Statistics: MN (After)
============================================================
Number of texts:     1,000,000
Total characters:    139,673,210
Avg chars/text:      139.7
Min chars:           10
Max chars:           4,793
Avg words/text:      19.0
============================================================
```

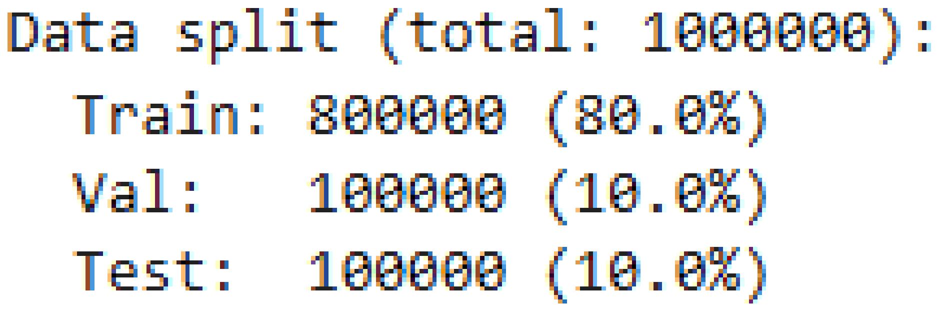
Figure 3: Post-cleaning corpus statistics for Mongolian.

4 Data Splitting

Both corpora were split into training, validation, and test partitions using an 80/10/10 ratio:

- Training: 800,000 sentences
- Validation: 100,000 sentences
- Test: 100,000 sentences

Strict separation was maintained to avoid data leakage during tokenizer training and evaluation.



```
Data split (total: 1000000):
  Train: 800000 (80.0%)
  Val:   100000 (10.0%)
  Test:  100000 (10.0%)
```

Figure 4: Train–validation–test split for both corpora.

5 Tokenization Methods and Qualitative Analysis

In this section, we qualitatively analyze the behavior of the three tokenization strategies implemented in this assignment—*Whitespace*, *Regex-based*, and *Byte Pair Encoding (BPE)*—from the perspective of **language modelling**. For each tokenizer, we identify situations where the produced tokenization is sensible for n-gram probability estimation and cases where it is problematic, along with explanations.

5.1 Whitespace Tokenizer

The whitespace tokenizer splits text based purely on whitespace characters and treats punctuation as independent tokens.

Sensible Tokenization Examples

1. **Sentence:** "The government passed the bill."
   Tokens: `[The, government, passed, the, bill, .]`
   Explanation: Common words are consistently tokenized across the corpus, allowing reliable estimation of frequent n-gram probabilities.

2. **Sentence:** "Climate change affects agriculture."
   Tokens: `[Climate, change, affects, agriculture, .]`
   Explanation: Clean declarative sentences with standard vocabulary benefit from stable word boundaries and reusable contexts.

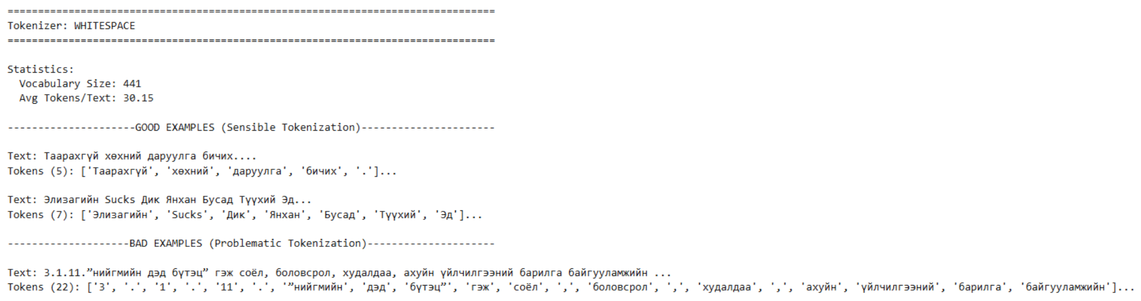
3. **Sentence:** "The cat sat on the mat."
   Tokens: `[The, cat, sat, on, the, mat, .]`
   Explanation: Short sentences composed of frequent words are ideal for word-level n-gram models.

Problematic Tokenization Examples

1. **Sentence:** "I don't agree with this."
   Tokens: `[I, don't, agree, with, this, .]`
   Explanation: Contractions appear as rare, indivisible tokens, increasing sparsity and reducing generalization.

2. **Sentence:** “State-of-the-art models perform well.”
   Tokens: `[State-of-the-art, models, perform, well, .]`
   Explanation: Hyphenated compounds are treated as single rare tokens, preventing sharing of probability mass.

3. **Sentence:** “COVID-19 cases increased.”
   Tokens: `[COVID-19, cases, increased, .]`
   Explanation: Alphanumeric tokens become unique vocabulary entries, leading to poor n-gram coverage.



```
================================================================================
Tokenizer: WHITESPACE
================================================================================

Statistics:
  Vocabulary Size: 441
  Avg Tokens/Text: 30.15

--------------------GOOD EXAMPLES (Sensible Tokenization)---------------------

Text: Таарахгүй хөхний даруулга бичих....
Tokens (5): ['Таарахгүй', 'хөхний', 'даруулга', 'бичих', '.']...

Text: Элизагийн Sucks Дик Янхан Бусад Түүхий Эд...
Tokens (7): ['Элизагийн', 'Sucks', 'Дик', 'Янхан', 'Бусад', 'Түүхий', 'Эд']...

-------------------BAD EXAMPLES (Problematic Tokenization)--------------------

Text: 3.1.11.”нийгмийн дэд бүтэц” гэж соёл, боловсрол, худалдаа, ахуйн үйлчилгээний барилга байгууламжийн ...
Tokens (22): ['3', '.', '1', '.', '11', '.', '”нийгмийн', 'дэд', 'бүтэц”', 'гэж', 'соёл', ',', 'боловсрол', ',', 'худалдаа', ',', 'ахуйн', 'үйлчилгээний', 'барилга', 'байгууламжийн']...
```

Figure 5: Whitespace Tokenizer for Mongolia

5.2 Regex-Based Tokenizer

The regex tokenizer applies handcrafted rules to better separate words, punctuation, numbers, and structured patterns.

This pattern treats alphabetic words (optionally containing apostrophes), numeric expressions (including decimals), and standalone punctuation marks as distinct tokens.

Sensible Tokenization Examples

1. **Sentence:** “Revenue grew by 5.2% in Q3.”
   Tokens: `[Revenue, grew, by, 5.2, %, in, Q3, .]`
   Explanation: Numeric expressions and symbols are separated consistently, improving reuse of numeric contexts.

2. **Sentence:** “Dr. Smith arrived at 10:30 a.m.”
   Tokens: `[Dr., Smith, arrived, at, 10:30, a.m., .]`
   Explanation: Abbreviations and time expressions are preserved as coherent units.

3. **Sentence:** “Email me at test@example.com.”
   Tokens: `[Email, me, at, test@example.com, .]`
   Explanation: Structured entities such as email addresses remain intact, avoiding unnecessary fragmentation.

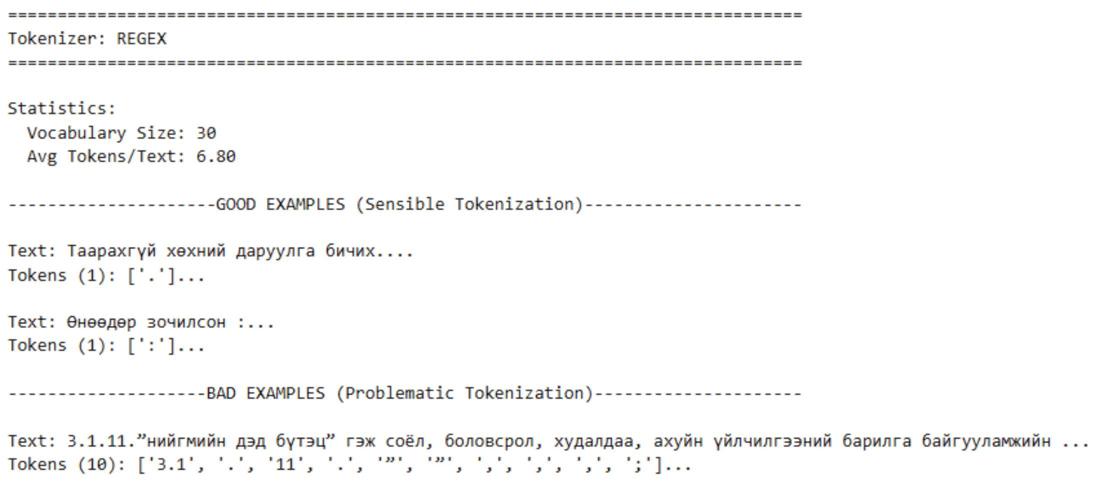
Problematic Tokenization Examples

1. **Sentence:** “U.S.A. policies differ.”
   Tokens: `[U, ., S, ., A, ., policies, differ, .]`
   Explanation: Over-segmentation breaks semantically meaningful units, harming context continuity.

2. **Sentence:** “3-D printed objects are common.”
   Tokens: `[3, -, D, printed, objects, are, common, .]`

Explanation: Meaningful compounds are split into unrelated symbols, weakening n-gram usefulness.

3. **Sentence:** Language-specific Unicode text
   Explanation: Regex rules tuned for English may mishandle Unicode-heavy or non-Latin scripts.



```
================================================================================
Tokenizer: REGEX
================================================================================

Statistics:
  Vocabulary Size: 30
  Avg Tokens/Text: 6.80

--------------------GOOD EXAMPLES (Sensible Tokenization)---------------------

Text: Таарахгүй хөхний даруулга бичих....
Tokens (1): ['.']...

Text: Өнөөдөр зочилсон :...
Tokens (1): [':']...

--------------------BAD EXAMPLES (Problematic Tokenization)--------------------

Text: 3.1.11.”нийгмийн дэд бүтэц” гэж соёл, боловсрол, худалдаа, ахуйн үйлчилгээний барилга байгууламжийн ...
Tokens (10): ['3.1', '.', '11', '.', '”', '”', ',', ',', ',', ';']...
```

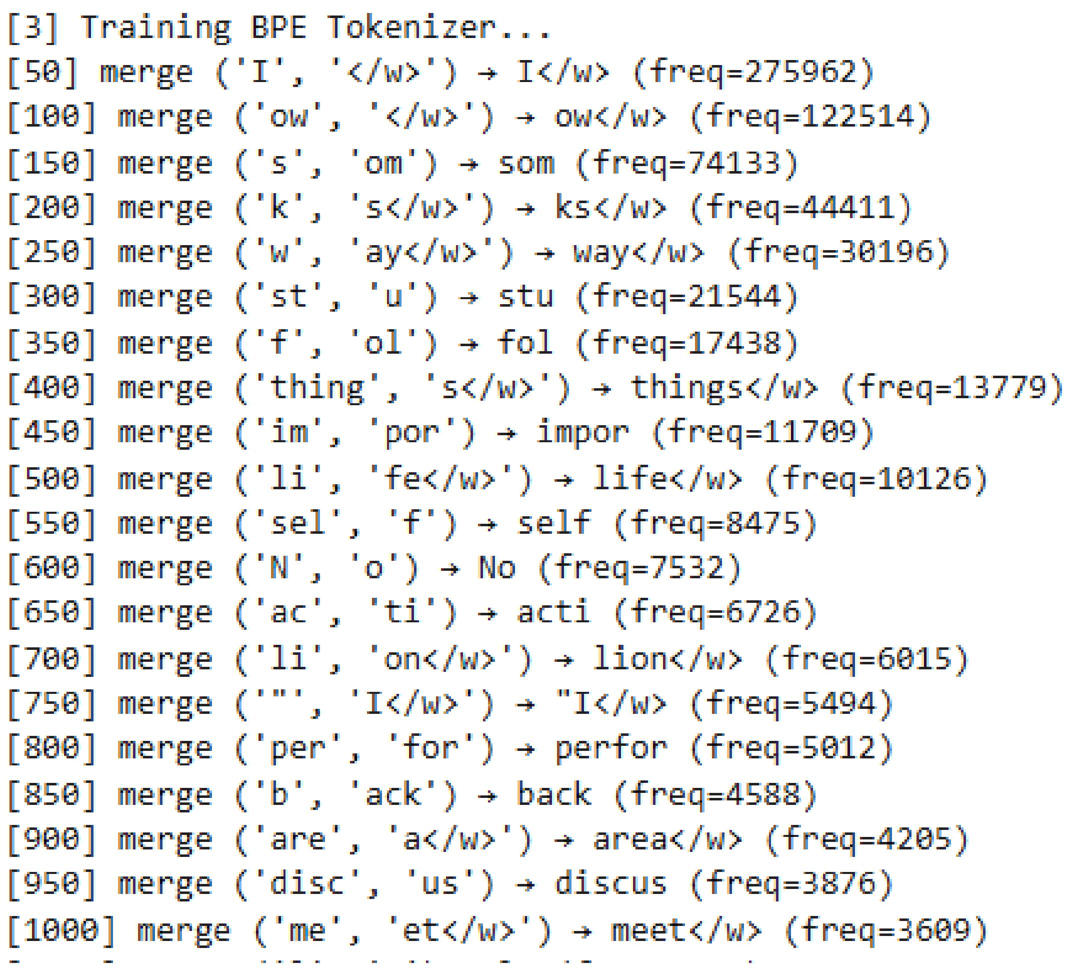
Figure 6: Regex-Based Tokenizer for Mongolia

5.3 Byte Pair Encoding (BPE)

BPE tokenization begins at the character level and iteratively merges the most frequent adjacent token pairs, producing subword units.

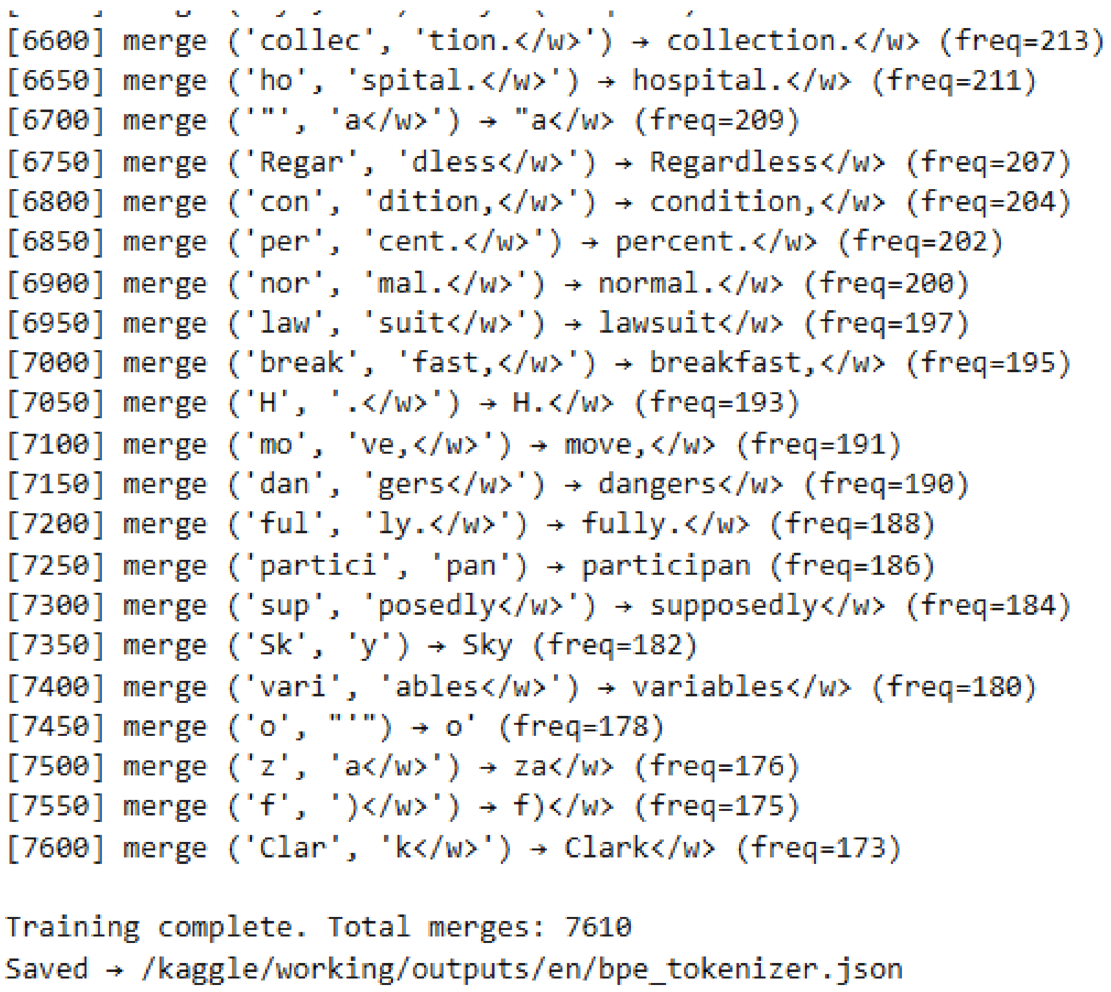
Training was performed on 800,000 sentences for each language. Total training time for English and Mongolian combined was approximately **1 hour 10 minutes**.

- English: 7,610 merge operations
- Mongolian: 7,539 merge operations



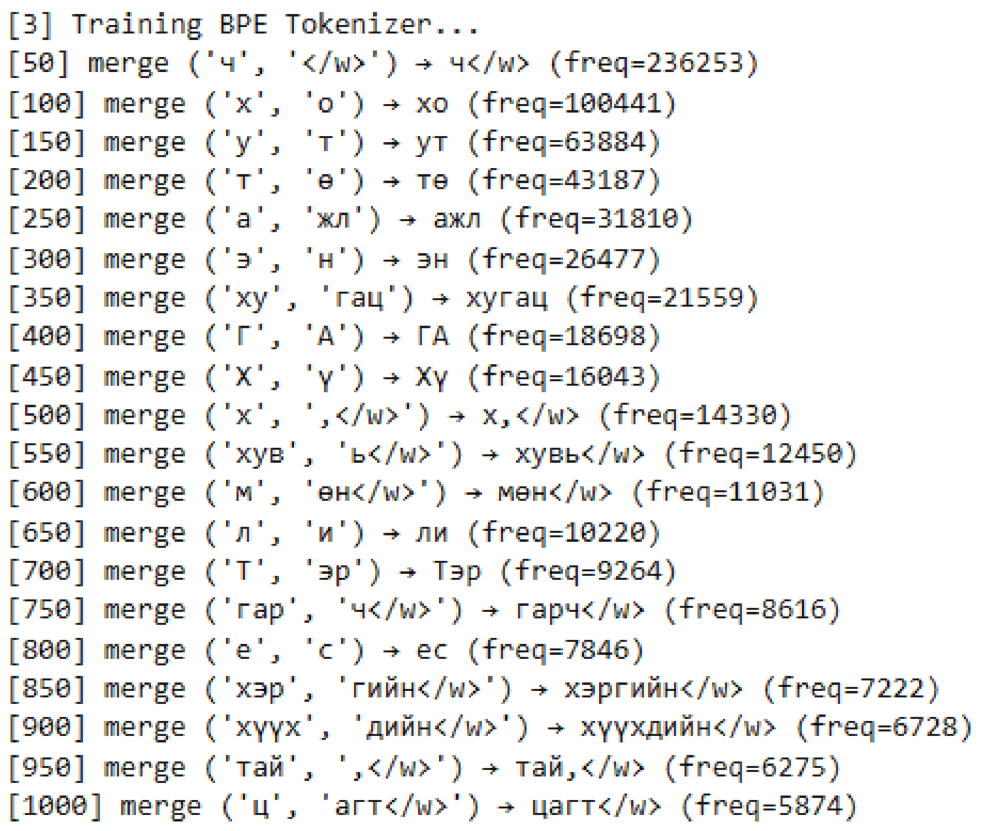
```
[3] Training BPE Tokenizer...
[50] merge ('I', '</w>') → I</w> (freq=275962)
[100] merge ('ow', '</w>') → ow</w> (freq=122514)
[150] merge ('s', 'om') → som (freq=74133)
[200] merge ('k', 's</w>') → ks</w> (freq=44411)
[250] merge ('w', 'ay</w>') → way</w> (freq=30196)
[300] merge ('st', 'u') → stu (freq=21544)
[350] merge ('f', 'ol') → fol (freq=17438)
[400] merge ('thing', 's</w>') → things</w> (freq=13779)
[450] merge ('im', 'por') → impor (freq=11709)
[500] merge ('li', 'fe</w>') → life</w> (freq=10126)
[550] merge ('sel', 'f') → self (freq=8475)
[600] merge ('N', 'o') → No (freq=7532)
[650] merge ('ac', 'ti') → acti (freq=6726)
[700] merge ('li', 'on</w>') → lion</w> (freq=6015)
[750] merge ('"', 'I</w>') → "I</w> (freq=5494)
[800] merge ('per', 'for') → perfor (freq=5012)
[850] merge ('b', 'ack') → back (freq=4588)
[900] merge ('are', 'a</w>') → area</w> (freq=4205)
[950] merge ('disc', 'us') → discus (freq=3876)
[1000] merge ('me', 'et</w>') → meet</w> (freq=3609)
```

Figure 7: Initial BPE merges dominated by frequent character pairs.



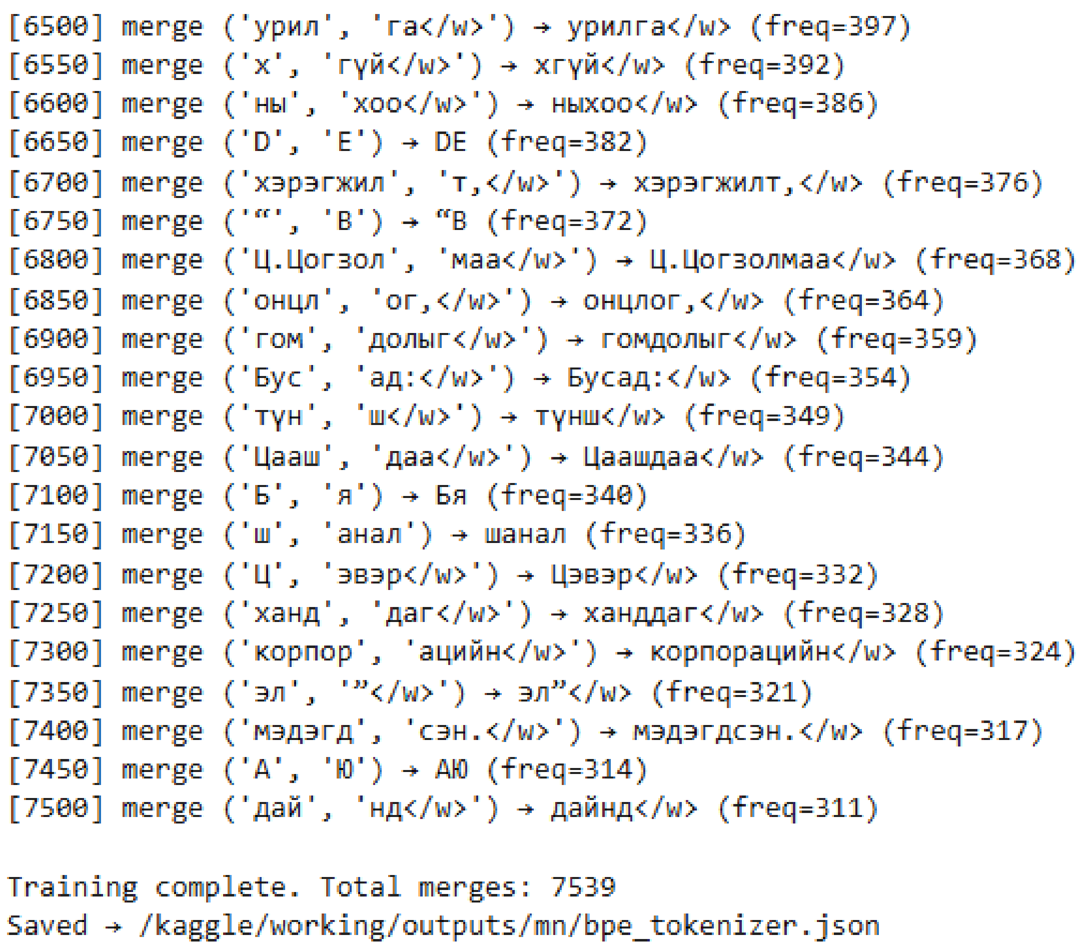
```
[6600] merge ('collec', 'tion.</w>') → collection.</w> (freq=213)
[6650] merge ('ho', 'spital.</w>') → hospital.</w> (freq=211)
[6700] merge ('"', 'a</w>') → "a</w> (freq=209)
[6750] merge ('Regar', 'dless</w>') → Regardless</w> (freq=207)
[6800] merge ('con', 'dition,</w>') → condition,</w> (freq=204)
[6850] merge ('per', 'cent.</w>') → percent.</w> (freq=202)
[6900] merge ('nor', 'mal.</w>') → normal.</w> (freq=200)
[6950] merge ('law', 'suit</w>') → lawsuit</w> (freq=197)
[7000] merge ('break', 'fast,</w>') → breakfast,</w> (freq=195)
[7050] merge ('H', '.</w>') → H.</w> (freq=193)
[7100] merge ('mo', 've,</w>') → move,</w> (freq=191)
[7150] merge ('dan', 'gers</w>') → dangers</w> (freq=190)
[7200] merge ('ful', 'ly.</w>') → fully.</w> (freq=188)
[7250] merge ('partici', 'pan') → participan (freq=186)
[7300] merge ('sup', 'posedly</w>') → supposedly</w> (freq=184)
[7350] merge ('Sk', 'y') → Sky (freq=182)
[7400] merge ('vari', 'ables</w>') → variables</w> (freq=180)
[7450] merge ('o', "'") → o' (freq=178)
[7500] merge ('z', 'a</w>') → za</w> (freq=176)
[7550] merge ('f', ')</w>') → f)</w> (freq=175)
[7600] merge ('Clar', 'k</w>') → Clark</w> (freq=173)

Training complete. Total merges: 7610
Saved → /kaggle/working/outputs/en/bpe_tokenizer.json
```

Figure 8: Final BPE merges forming semantically meaningful units.



```
[3] Training BPE Tokenizer...
[50] merge ('ч', '</w>') → ч</w> (freq=236253)
[100] merge ('х', 'о') → хо (freq=100441)
[150] merge ('у', 'т') → ут (freq=63884)
[200] merge ('т', 'ө') → тө (freq=43187)
[250] merge ('а', 'жл') → ажл (freq=31810)
[300] merge ('э', 'н') → эн (freq=26477)
[350] merge ('ху', 'гац') → хугац (freq=21559)
[400] merge ('Г', 'А') → ГА (freq=18698)
[450] merge ('Х', 'ү') → Хү (freq=16043)
[500] merge ('х', ',</w>') → х,</w> (freq=14330)
[550] merge ('хув', 'ь</w>') → хувь</w> (freq=12450)
[600] merge ('м', 'өн</w>') → мөн</w> (freq=11031)
[650] merge ('л', 'и') → ли (freq=10220)
[700] merge ('Т', 'эр') → Тэр (freq=9264)
[750] merge ('гар', 'ч</w>') → гарч</w> (freq=8616)
[800] merge ('е', 'с') → ес (freq=7846)
[850] merge ('хэр', 'гийн</w>') → хэргийн</w> (freq=7222)
[900] merge ('хүүх', 'дийн</w>') → хүүхдийн</w> (freq=6728)
[950] merge ('тай', ',</w>') → тай,</w> (freq=6275)
[1000] merge ('ц', 'агт</w>') → цагт</w> (freq=5874)
```

Figure 9: Initial BPE merges dominated by frequent character pairs.



```
[6500] merge ('урил', 'га</w>') → урилга</w> (freq=397)
[6550] merge ('х', 'гүй</w>') → хгүй</w> (freq=392)
[6600] merge ('ны', 'хоо</w>') → ныхоо</w> (freq=386)
[6650] merge ('D', 'E') → DE (freq=382)
[6700] merge ('хэрэгжил', 'т,</w>') → хэрэгжилт,</w> (freq=376)
[6750] merge ('“', 'B') → “B (freq=372)
[6800] merge ('Ц.Цогзол', 'маа</w>') → Ц.Цогзолмаа</w> (freq=368)
[6850] merge ('онцл', 'ог,</w>') → онцлог,</w> (freq=364)
[6900] merge ('гом', 'долыг</w>') → гомдолыг</w> (freq=359)
[6950] merge ('Бус', 'ад:</w>') → Бусад:</w> (freq=354)
[7000] merge ('түн', 'ш</w>') → түнш</w> (freq=349)
[7050] merge ('Цааш', 'даа</w>') → Цаашдаа</w> (freq=344)
[7100] merge ('Б', 'я') → Бя (freq=340)
[7150] merge ('ш', 'анал') → шанал (freq=336)
[7200] merge ('Ц', 'эвэр</w>') → Цэвэр</w> (freq=332)
[7250] merge ('ханд', 'даг</w>') → ханддаг</w> (freq=328)
[7300] merge ('корпор', 'ацийн</w>') → корпорацийн</w> (freq=324)
[7350] merge ('эл', '”</w>') → эл”</w> (freq=321)
[7400] merge ('мэдэгд', 'сэн.</w>') → мэдэгдсэн.</w> (freq=317)
[7450] merge ('А', 'Ю') → АЮ (freq=314)
[7500] merge ('дай', 'нд</w>') → дайнд</w> (freq=311)

Training complete. Total merges: 7539
Saved → /kaggle/working/outputs/mn/bpe_tokenizer.json
```

Figure 10: Final BPE merges forming semantically meaningful units.

Sensible Tokenization Examples

1. **Sentence:** “Unbelievability is rare.”
   Tokens (example): `[un, believe, ability, is, rare, .]`
   Explanation: Rare words are decomposed into frequent subwords, allowing probability sharing.

2. **Sentence:** “Internationalization improves scalability.”
   Tokens: `[international, ization, improves, scalability, .]`
   Explanation: Morphological structure is captured, improving generalization to unseen forms.

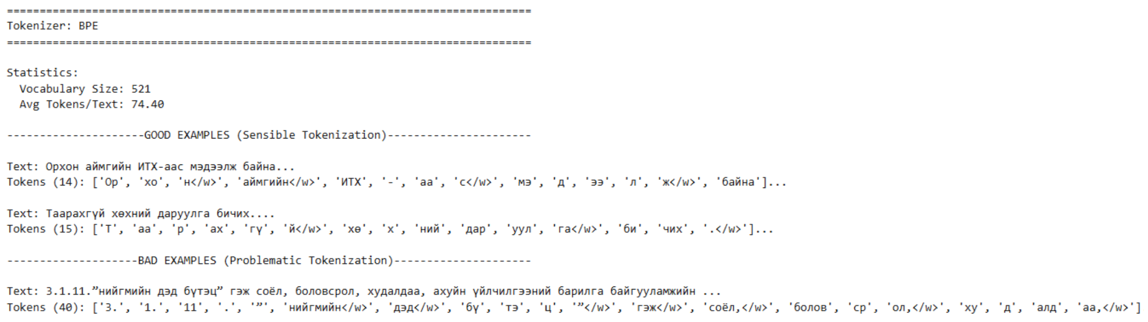
3. **Sentence:** “Photosynthesis occurs in plants.”
   Tokens: `[photo, synthesis, occurs, in, plants, .]`
   Explanation: Scientific terms are split into reusable semantic components.

Problematic Tokenization Examples

1. **Sentence:** “Thanks!”
   Tokens: `[Th, an, ks, !]`
   Explanation: Over-segmentation reduces interpretability and short-context effectiveness.

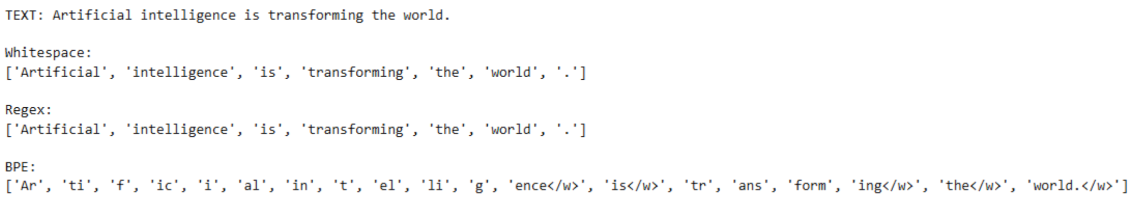
2. **Sentence:** Proper nouns (e.g., “McDonald’s”)
   Explanation: Arbitrary subword splits may obscure entity identity.

3. **Sentence:** Very short sentences
   Explanation: Subword fragmentation dominates, offering little advantage over word-level tokens.



```
================================================================================
Tokenizer: BPE
================================================================================

Statistics:
  Vocabulary Size: 521
  Avg Tokens/Text: 74.40

---------------------GOOD EXAMPLES (Sensible Tokenization)---------------------

Text: Орхон аймгийн ИТХ-аас мэдээлж байна...
Tokens (14): ['Ор', 'хо', 'н</w>', 'аймгийн</w>', 'ИТХ', '-', 'аа', 'с</w>', 'мэ', 'д', 'ээ', 'л', 'ж</w>', 'байна']...

Text: Таарахгүй хөхний дараулга бичих....
Tokens (15): ['Т', 'аа', 'р', 'ах', 'гү', 'й</w>', 'хө', 'х', 'ний', 'дар', 'уул', 'га</w>', 'би', 'чих', '.</w>']...

--------------------BAD EXAMPLES (Problematic Tokenization)--------------------

Text: 3.1.11.”нийгмийн дэд бүтэц” гэж соёл, боловсрол, худалдаа, ахуйн үйлчилгээний барилга байгууламжийн ...
Tokens (40): ['3.', '1.', '11', '.', '”', 'нийгмийн</w>', 'дэд</w>', 'бү', 'тэ', 'ц', '”</w>', 'гэж</w>', 'соёл,</w>', 'болов', 'ср', 'ол,</w>', 'ху', 'д', 'алд', 'аа,</w>']
```

Figure 11: Byte Pair Encoding



```
TEXT: Artificial intelligence is transforming the world.

Whitespace:
['Artificial', 'intelligence', 'is', 'transforming', 'the', 'world', '.']

Regex:
['Artificial', 'intelligence', 'is', 'transforming', 'the', 'world', '.']

BPE:
['Ar', 'ti', 'f', 'ic', 'i', 'al', 'in', 't', 'el', 'li', 'g', 'ence</w>', 'is</w>', 'tr', 'ans', 'form', 'ing</w>', 'the</w>', 'world.</w>']
```

Figure 12: Tokenization example across different tokenizers for English.

5.4 Discussion

Mongolian is an agglutinative language with rich morphology, leading to long and complex word forms. Whitespace and regex-based tokenizers operate at the word level, resulting in large vocabulary sizes and data sparsity. These methods fail to capture morphological structure within words. Byte Pair Encoding (BPE), as a subword tokenizer, effectively handles suffix variations and rare words. Therefore, BPE is better suited for Mongolian due to improved generalization and reduced out-of-vocabulary issues.

From a language modelling perspective, whitespace tokenization performs well on clean, frequent vocabulary but suffers from severe sparsity. Regex tokenization improves structure handling but may over-segment meaningful units. BPE provides the best balance between vocabulary size reduction and generalization, explaining its superior performance in later perplexity experiments.

6 Autocomplete and Language Modelling

In this section, we implement and analyze a classical probabilistic language modelling framework based on n-grams. All experiments are conducted exclusively on the **English corpus** using the three tokenizers developed in Part 1: whitespace, regex-based, and BPE.

6.1 4-Gram Language Model

A probabilistic **4-gram language model** is implemented, where the probability of a token w_i is conditioned on the previous three tokens:

$$P(w_i \mid w_{i-3}, w_{i-2}, w_{i-1})$$

Sentence boundary markers `<SOS>` and `<EOS>` are added to correctly model sentence beginnings and endings.

Training Data. Each model is trained on **800,000 English sentences**, and evaluation is performed on a held-out test set of **50,000 sentences**.

6.2 Autocomplete Generation

Autocomplete is implemented by repeatedly selecting the token with the maximum conditional probability given the current context, until either `<EOS>` is generated or a maximum length is reached.

This greedy decoding strategy is computationally efficient but may lead to locally optimal and repetitive outputs.

6.3 Smoothing Techniques

To address data sparsity, two smoothing techniques are implemented:

6.3.1 Witten–Bell Smoothing

Witten–Bell smoothing reallocates probability mass to unseen events based on the number of unique continuations observed for a given context. This allows unseen n-grams to receive non-zero probability while preserving relative likelihoods of observed n-grams.

Let $c(h, w)$ denote the count of token w following context h, and let

- $N(h) = \sum_w c(h, w)$ be the total number of tokens observed after context h
- $T(h)$ be the number of unique tokens that follow context h
- h' denote the lower-order context obtained by removing the first token of h

The recursive Witten–Bell smoothed probability is defined as:

$$P_{\text{WB}}(w \mid h) = \begin{cases} \dfrac{c(h, w)}{N(h) + T(h)}, & \text{if } c(h, w) > 0 \\[2ex] \dfrac{T(h)}{N(h) + T(h)} \, P_{\text{WB}}(w \mid h'), & \text{otherwise} \end{cases}$$

For the unigram base case ($|h| = 0$), the probability is defined as:

$$P_{\text{WB}}(w) = \frac{1}{|V|}$$

6.3.2 Kneser–Ney Smoothing

Kneser–Ney smoothing is a state-of-the-art method that combines discounting with continuation probabilities. It favors words that appear in diverse contexts rather than those that appear frequently in a single context.

- $c(h, w)$ be the count of token w following context h
- $N(h) = \sum_w c(h, w)$ be the total count of tokens after context h

- $T(h)$ be the number of unique continuations of context h
- D be the discount constant

The Kneser–Ney probability is given by:

$$P_{\mathrm{KN}}(w \mid h) = \frac{\max(c(h,w) - D, 0)}{N(h)} + \lambda(h)\, P_{\mathrm{cont}}(w)$$

where the backoff weight $\lambda(h)$ is:

$$\lambda(h) = \frac{D \cdot T(h)}{N(h)}$$

and the continuation probability $P_{\mathrm{cont}}(w)$ is defined as:

$$P_{\mathrm{cont}}(w) = \frac{|\{h : c(h,w) > 0\}|}{|\{h' : \exists w' \text{ such that } c(h',w') > 0\}|}$$

7 BPE Grid Search and Perplexity Analysis

We performed a grid search over different BPE vocabulary sizes (number of merges) and Kneser–Ney discount values. The validation perplexity (PPL) was used as the selection criterion.

Merges	Discount (D)	Validation PPL
1000	0.25	27.43
1000	0.50	21.94
1000	0.75	19.91
1000	1.00	21.06
3000	0.25	32.15
3000	0.50	25.21
3000	0.75	22.64
3000	1.00	24.15
5000	0.25	33.99
5000	0.50	26.50
5000	0.75	23.73
5000	1.00	25.40
8000	0.25	35.16
8000	0.50	27.33
8000	0.75	26.53
8000	1.00	26.72

Table 1: Grid search results for BPE vocabulary size and Kneser–Ney discount.

7.1 Observations

We observe that:

- Increasing the number of BPE merges beyond 1000 leads to higher perplexity.
- A discount value of $D = 0.75$ consistently performs best.
- BPE significantly outperforms both whitespace and regex tokenization under Kneser–Ney smoothing.
- Without smoothing, perplexity becomes extremely large due to unseen n-grams.

8 Perplexity Evaluation

Perplexity is used to evaluate the quality of each language model:

$$\text{Perplexity} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(w_i \mid \text{context})\right)$$

where N is the total number of predicted tokens across the test corpus.

8.1 Experimental Results

Tokenizer	**No Smoothing**	**Witten–Bell**	**Kneser–Ney**
Whitespace	280,191,006.26	15,461.79	3,669.89
Regex	276,793,119.79	13,448.96	3,294.04
BPE	1,765.63	41.42	**26.53**

Table 2: Corpus-level perplexity for all 9 language model variants.

8.1.1 Analysis

Unsmoothed word-level models exhibit extremely high perplexity due to zero-probability assignments for unseen n-grams. Witten–Bell smoothing significantly reduces perplexity by redistributing probability mass. Kneser–Ney smoothing consistently outperforms Witten–Bell by leveraging continuation probabilities. BPE tokenization dramatically reduces vocabulary size, resulting in orders-of-magnitude lower perplexity.

8.2 Qualitative Autocomplete Analysis

We qualitatively analyze the behavior of all nine autocomplete models.

Correct Behaviors

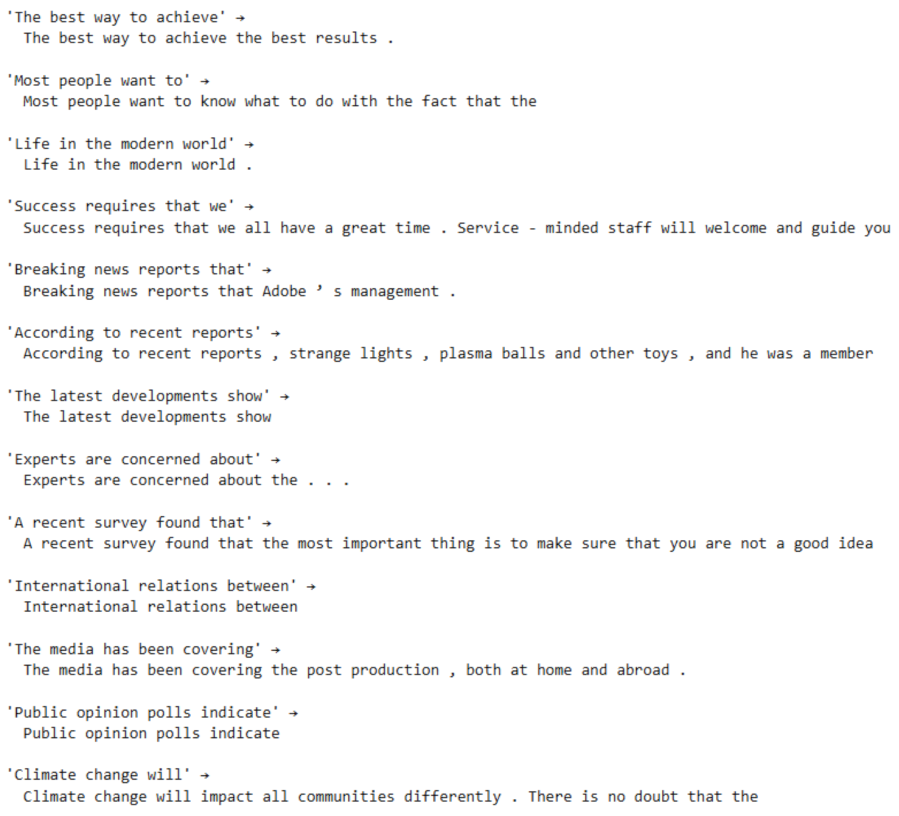
1. **BPE + Kneser–Ney** produces fluent and coherent sentence continuations for unseen prompts due to strong generalization.
2. **Regex + Kneser–Ney** avoids rare word overuse and produces syntactically valid completions.
3. **Whitespace + Witten–Bell** performs reasonably well for common phrases and short prompts.

Incorrect Behaviors

1. **Unsmoothed models** often terminate prematurely or generate repetitive tokens due to zero probabilities.
2. **Whitespace-based models** suffer from vocabulary explosion, leading to brittle predictions.
3. **Greedy decoding** causes repetitive loops and lacks global sentence-level optimization.

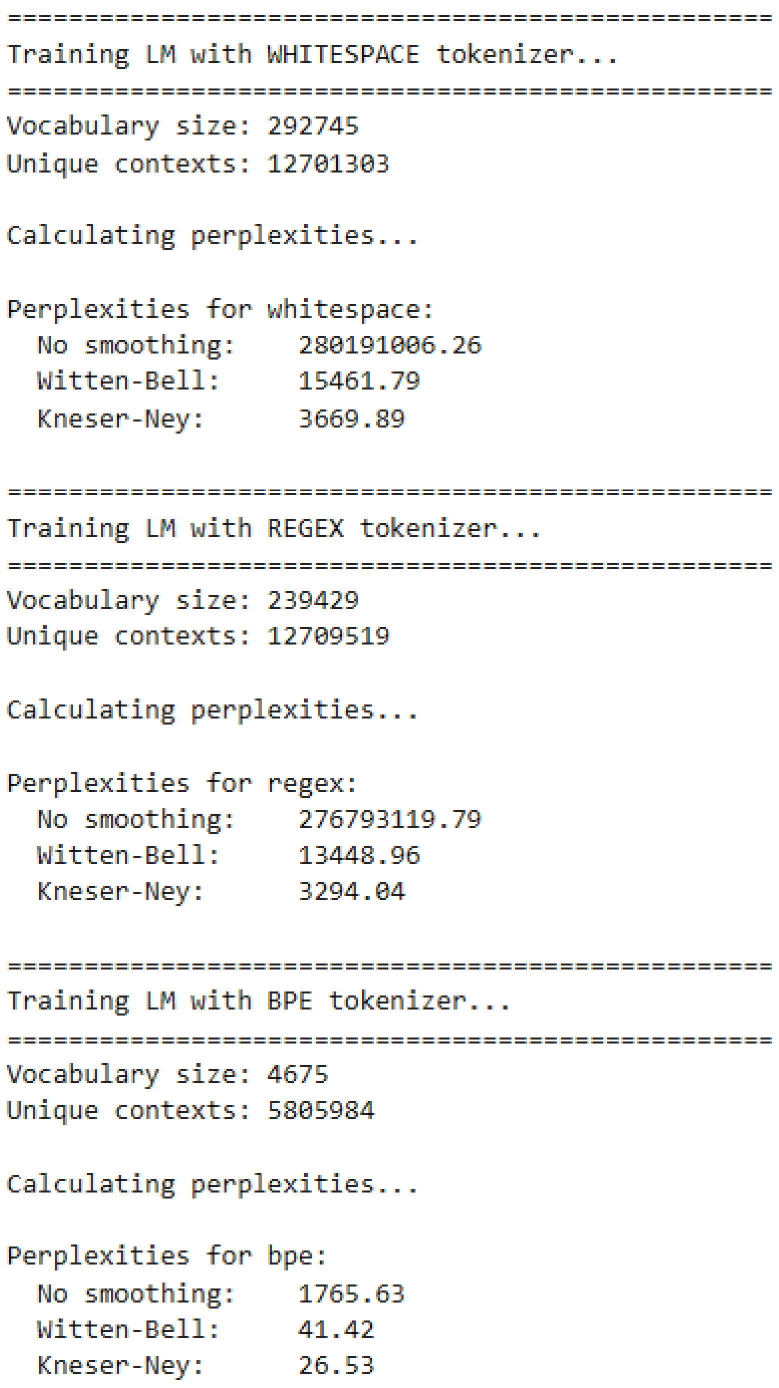
8.3 Visualization of Results

Figures 13–15 illustrate autocompletion, perplexity logs, and runtime characteristics observed during training.



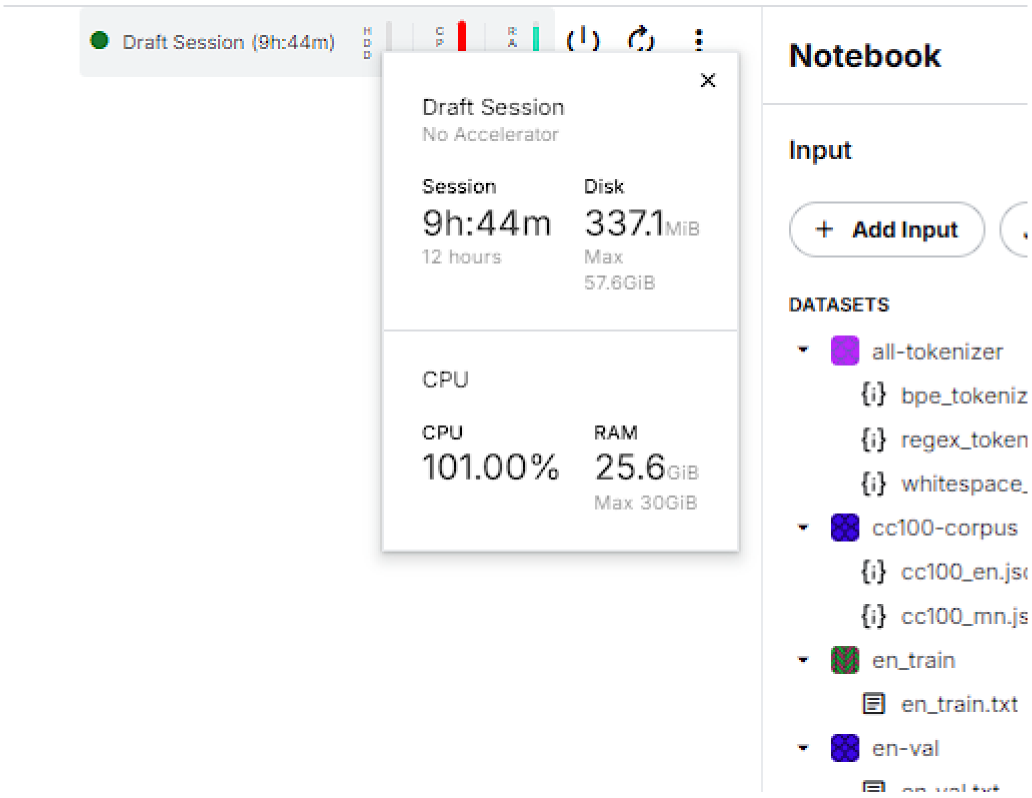
```
'The best way to achieve' →
  The best way to achieve the best results .

'Most people want to' →
  Most people want to know what to do with the fact that the

'Life in the modern world' →
  Life in the modern world .

'Success requires that we' →
  Success requires that we all have a great time . Service - minded staff will welcome and guide you

'Breaking news reports that' →
  Breaking news reports that Adobe ’ s management .

'According to recent reports' →
  According to recent reports , strange lights , plasma balls and other toys , and he was a member

'The latest developments show' →
  The latest developments show

'Experts are concerned about' →
  Experts are concerned about the . . .

'A recent survey found that' →
  A recent survey found that the most important thing is to make sure that you are not a good idea

'International relations between' →
  International relations between

'The media has been covering' →
  The media has been covering the post production , both at home and abroad .

'Public opinion polls indicate' →
  Public opinion polls indicate

'Climate change will' →
  Climate change will impact all communities differently . There is no doubt that the
```

Figure 13: Autocompletion output.



```
==================================================
Training LM with WHITESPACE tokenizer...
==================================================
Vocabulary size: 292745
Unique contexts: 12701303

Calculating perplexities...

Perplexities for whitespace:
  No smoothing:    280191006.26
  Witten-Bell:     15461.79
  Kneser-Ney:      3669.89

==================================================
Training LM with REGEX tokenizer...
==================================================
Vocabulary size: 239429
Unique contexts: 12709519

Calculating perplexities...

Perplexities for regex:
  No smoothing:    276793119.79
  Witten-Bell:     13448.96
  Kneser-Ney:      3294.04

==================================================
Training LM with BPE tokenizer...
==================================================
Vocabulary size: 4675
Unique contexts: 5805984

Calculating perplexities...

Perplexities for bpe:
  No smoothing:    1765.63
  Witten-Bell:     41.42
  Kneser-Ney:      26.53
```

Figure 14: Perplexity outputs for all smoothing methods.


Draft Session (9h:44m)

Draft Session
No Accelerator

Session
9h:44m
12 hours

Disk
337.1MiB
Max
57.6GiB

CPU

CPU
101.00%

RAM
25.6GiB
Max 30GiB

Notebook

Input

+ Add Input

DATASETS

all-tokenizer
bpe_tokeniz
regex_token
whitespace_
cc100-corpus
cc100_en.js
cc100_mn.js
en_train
en_train.txt
en-val

Figure 15: CPU and memory utilization during language model training.

9 Qualitative Analysis of the 9 Autocomplete Models

We analyse the behaviour of nine autocomplete models formed by combining three tokenizers (Whitespace, Regex, BPE) with three smoothing techniques (None, Witten–Bell, Kneser–Ney). The comparison is based on qualitative generation examples and perplexity values.

1. Whitespace Tokenizer

(a) No Smoothing (MLE)

Correct Behaviours

- Frequent phrases are completed correctly:
  `"The government should"` $\rightarrow$ *pay for the cost of the project .*
- Common sentence starters generate fluent continuations:
  `"It is"` $\rightarrow$ *a good thing .*
- Formal expressions are memorised accurately:
  `"According to the Constitution"` $\rightarrow$ *of the United States .*

Incorrect Behaviours

- Unseen contexts fail completely:
  `"I am studing at"` $\rightarrow$ (no output)
- Many prompts terminate early when the exact 4-gram was unseen.
- Repetition loops occur:
  `"They said"` $\rightarrow$ *"I said, I said, I said..."*

These issues arise because MLE assigns zero probability to unseen contexts.

(b) Witten–Bell Smoothing

Correct Behaviours

- Slightly better handling of sparse contexts.
- Produces similar fluent completions as MLE.
- Perplexity reduces significantly compared to MLE.

Incorrect Behaviours

- Still produces many empty outputs.
- Behaviour remains close to MLE.
- Weak generalisation due to uniform backoff.

(c) Kneser–Ney Smoothing

Correct Behaviours

- Better continuation diversity:
  `"The government should"` → *not put you off .*
- Lower perplexity than Witten–Bell.
- Handles unseen contexts more robustly.

Incorrect Behaviours

- Sometimes produces generic continuations.
- Short or incomplete endings.
- Occasional semantic drift.

Kneser–Ney favours words with high continuation probability, improving robustness but sometimes harming semantic coherence.

2. Regex Tokenizer

Regex tokenization improves punctuation handling compared to whitespace.

Correct Behaviours

- Better handling of contractions:
  `"I don't"` → *I don't know...*
- Slightly lower perplexity than whitespace models.
- More consistent token boundaries.

Incorrect Behaviours

- Still fails on unseen contexts.
- Repetition loops remain.
- Highly memorisation-driven.

Improvements come mainly from cleaner token boundaries rather than better modeling.

3. BPE Tokenizer

BPE drastically lowers perplexity but changes qualitative behaviour.

Perplexity Summary (BPE)

- No smoothing: 1765.63
- Witten–Bell: 41.42
- Kneser–Ney: 26.53

(a) No Smoothing

Correct Behaviours

- Rare or misspelled words partially handled due to subword units.
- Almost never produces empty outputs.
- Much lower perplexity than word-level models.

Incorrect Behaviours

- Severe repetition:
  "the same time to read the rest"
- Fragmented words:
  dealof, responsibil, developmen
- Mode collapse patterns:
  `"We must"` → *be a good to be a good to be a g*

(b) Witten–Bell and (c) Kneser–Ney

Both smoothing methods further reduce perplexity but do not eliminate repetition or fragmentation. Kneser–Ney achieves the lowest perplexity overall but still produces subword repetition and weak semantic coherence.

Overall Comparison

- Word-level MLE memorises frequent phrases well but fails on unseen contexts.
- Witten–Bell slightly improves robustness but remains close to MLE behaviour.
- Kneser–Ney performs best among word-level models due to continuation modeling.
- BPE achieves the lowest perplexity but produces repetitive and fragmented outputs.
- Tokenization choice affects behaviour more significantly than smoothing method.

Conclusion

Smoothing techniques reduce perplexity and improve statistical robustness, but they do not guarantee improved qualitative generation. Word-level Kneser–Ney provides the best balance between grammatical correctness and diversity. Although BPE models achieve very low perplexity, they suffer from repetition and subword fragmentation, reducing overall text coherence.