Assignment 1: Tokenization and Language Modeling

Introduction to NLP - Spring '26

Ankit Kumar
Roll Number: 2025201022

February 11, 2026

---

1 Introduction

This report presents the implementation and analysis of tokenization methods and N-gram language models with various smoothing techniques for English and Mongolian corpora from the CC-100 dataset.

2 Task 1: Tokenization

2.1 Data Cleaning and Partitioning (Task 1.1)

2.1.1 Cleaning Steps Implemented:

- Removal of URLs using regex pattern
- Normalization of Unicode quotes and apostrophes
- Removal of non-breaking spaces
- Collapsing multiple whitespace characters
- Stripping leading/trailing whitespace

2.1.2 Data Partitioning Ratios:

- **Training:** 80%
- **Validation:** 10%
- **Testing:** 10%

2.2 Tokenizer Implementation (Task 1.2)

Whitespace Tokenizer
Description: Splits text on whitespace characters only.
Simplifying Assumptions:

- Punctuation remains attached to adjacent words if there is no space between them.
- No special handling of contractions (e.g., "don't" stays as one token).

Regex Tokenizer
Description: Uses regular expression to separate words, numbers, and punctuation.
Pattern: `[^\W \d _]+|\d +|[^\w \s ]`
Simplifying Assumptions:

- Each punctuation mark is treated as a separate token
- Numbers are kept as single tokens
- Unicode word characters supported

BPE Tokenizer
Description: Byte Pair Encoding learns subword units through iterative merging.
Parameters: 1000 merge operations
Simplifying Assumptions:

- Words initially split by whitespace
- End-of-word marker `</w>` added to each word
- Character-level fallback for unknown words

2.3 Tokenization Analysis (Task 1.3)

2.3.1 Sensible Tokenizations English corpus (3 examples)

Whitespace Tokenizer

- **Example 1:**
  - **Sentence:** 'The cat sat'
  - **Output:** `['The', 'cat', 'sat']`
  - **Analysis:** This tokenization preserves complete words and semantic meaning, allowing the model to learn strong word-level associations such as "cat $\rightarrow$ sat." It supports reliable next-word prediction because these common word sequences frequently appear in training data.
- **Example 2:**
  - **Sentence:** 'Hello, world!'
  - **Output:** `['Hello,', 'world!']`
  - **Analysis:** Although punctuation remains attached, the core words are preserved, enabling the model to capture meaningful co-occurrence patterns. It is still useful for predicting sentence-level structures like greetings despite minor vocabulary inflation.
- **Example 3:**
  - **Sentence:** 'I love NLP'
  - **Output:** `['I', 'love', 'NLP']`
  - **Analysis:** Each token represents a clear linguistic unit, making probability estimation straightforward. Frequent phrases like "I love" help the model generalize well and improve prediction accuracy.

Regex Tokenizer

- **Example 1:**
  - **Sentence:** 'hello!'
  - **Output:** `['hello', '!']`
  - **Analysis:** Separating punctuation helps the model explicitly learn that exclamation marks often follow greetings. This improves sentence boundary prediction and reduces noise caused by punctuation-attached words.

- **Example 2:**
  - **Sentence:** 'Price is $5'
  - **Output:** `['price', 'is', '$', '5']`
  - **Analysis:** Breaking currency symbols from numbers allows the model to learn structured patterns in financial expressions. It improves generalization since "$" and numeric values can appear in multiple contexts.

- **Example 3:**
  - **Sentence:** 'Are you sure?'
  - **Output:** `['are', 'you', 'sure', '?']`
  - **Analysis:** The tokenizer preserves word meaning while isolating the question mark, helping the model recognize interrogative sentence patterns. This supports better prediction of conversational structures.

BPE Tokenizer

- **Example 1:**
  - **Sentence:** 'running'
  - **Output:** `['run', 'ing']`
  - **Analysis:** Splitting into "run" and "ing" captures morphological structure, allowing the model to generalize across related forms like "walking" or "playing." This reduces sparsity and improves prediction reliability.

- **Example 2:**
  - **Sentence:** 'unhappiness'
  - **Output:** `['un', 'happi', 'ness']`
  - **Analysis:** The subwords represent prefix, root, and suffix components, enabling the model to understand negation and noun formation. Such decomposition improves handling of rare or unseen words.

- **Example 3:**
  - **Sentence:** 'International'
  - **Output:** `['inter', 'national']`
  - **Analysis:** Dividing the word into frequent subunits reduces vocabulary size while preserving semantic cues. These reusable components strengthen probability estimates across different contexts.

2.3.2 Non-Sensible Tokenizations English corpus (3 examples)

Whitespace Tokenizer

- **Example 1:**
  - **Sentence:** 'hello!'
  - **Output:** `['hello!']`
  - **Problem:** Attaching punctuation to the word creates a new token that rarely appears elsewhere, increasing vocabulary size and sparsity while weakening generalization.

- **Example 2:**
  - **Sentence:** [hello , world!]
  - **Output:** `['hello', ',', 'world!']`
  - **Problem:** Even when space is present before a comma, the exclamation mark at the end of the sentence remains attached to “world”, creating a rare token that hinders learning.

- **Example 3:**
  - **Sentence:** [don’t stop]
  - **Output:** `["don't", "stop"]`
  - **Problem:** Keeping the contraction as a single token prevents the model from learning relationships between “do” and “not,” reducing its ability to generalize across similar constructions.

Regex Tokenizer

- **Example 1:**
  - **Sentence:** ’U.S.A.’
  - **Output:** `['u', '.', 's', '.', 'a']`
  - **Problem:** Splitting an abbreviation into individual letters destroys its semantic identity, producing tokens that rarely occur independently and harming probability estimation.

- **Example 2:**
  - **Sentence:** ’C++’
  - **Output:** `['c', '+', '+']`
  - **Problem:** Breaking a technical term into symbols fragments a meaningful unit, making it harder for the model to learn domain-specific vocabulary.

- **Example 3:**
  - **Sentence:** ’3.14’
  - **Output:** `['3', '.', '14']`
  - **Problem:** Separating a decimal number disrupts a single numeric concept, potentially confusing the model when learning patterns involving measurements or prices.

BPE Tokenizer

- **Example 1:**
  - **Sentence:** ’the’
  - **Output:** `['t', 'he']`
  - **Problem:** Over-segmentation of a very frequent word increases sequence length unnecessarily and weakens fluency during generation.

- **Example 2:**
  - **Sentence:** ’market’
  - **Output:** `['mar', 'ke', 't']`
  - **Problem:** Fragmenting a common word into multiple subwords reduces interpretability and may dilute meaningful statistical patterns.

- **Example 3:**
  - **Sentence:** ’$63’
  - **Output:** `['$', '6', '3']`
  - **Problem:** Splitting the currency amount into small units breaks a coherent numerical expression, making it harder for the model to learn financial formats.

2.3.3 Sensible Tokenizations Mongolian corpus (3 examples)

Whitespace Tokenizer

- **Example 1:**
  - **Sentence:** ’Би ном уншиж байна’
  - **Output:** `['Би', 'ном', 'уншиж', 'байна']`
  - **Analysis:** Preserves natural word boundaries and meaning. Each token corresponds to a grammatical unit, helping the language model learn word order and predict context effectively.

- **Example 2:**
  - **Sentence:** ’Тэр сургуульд явсан’
  - **Output:** `['Тэр', 'сургуульд', 'явсан']`
  - **Analysis:** Correctly separates subject, location, and verb. This improves syntactic understanding and reduces ambiguity during probability estimation.

- **Example 3:**
  - **Sentence:** ’Өнөөдөр хүйтэн байна’
  - **Output:** `['өнөөдөр', 'хүйтэн', 'байна']`
  - **Analysis:** Maintains semantic clarity by keeping descriptive words separate. Useful for modeling sentence structure and capturing contextual relationships.

Regex Tokenizer

- **Example 1:**
  - **Sentence:** ’Сайн байна уу?’
  - **Output:** `['Сайн', 'байна', 'уу', '?']`
  - **Analysis:** Separates punctuation from words, allowing the model to treat “?” as a sentence-ending signal. This improves understanding of sentence types and structure.

- **Example 2:**
  - **Sentence:** ’Тэр ирлээ!’
  - **Output:** `['Тэр', 'ирлээ', '!']`
  - **Analysis:** By isolating the exclamation mark, the tokenizer captures emotional or emphatic context, which can help in predicting discourse patterns.

- **Example 3:**
  - **Sentence:** ’Үнэ 500₮.’
  - **Output:** `['Үнэ', '500', '₮', '.']`
  - **Analysis:** Splitting numbers, currency symbols, and punctuation creates structured tokens that improve numerical understanding and reduce vocabulary noise.

BPE Tokenizer

- **Example 1:**
  - **Sentence:** ’сургуулиудад’
  - **Output:** `['сургууль', 'ууд', 'ад']`
  - **Analysis:** Breaks the word into meaningful morphemes (root + suffixes), enabling the model to generalize across inflected forms in this morphologically rich language.

- **Example 2:**
  - **Sentence:** ’уншсан’
  - **Output:** `['унш', 'сан']`

- **Analysis:** Captures the verb root and tense marker separately, reducing sparsity and helping the model recognize related word variations.

- **Example 3:**
  - **Sentence:** ’хүүхдүүд’
  - **Output:** `['хүүхэд', 'үүд']`
  - **Analysis:** Identifies the base noun and plural suffix, improving vocabulary efficiency while preserving semantic meaning.

2.3.4 Non-Sensible Tokenizations Mongolian corpus (3 examples)

Whitespace Tokenizer

- **Example 1:**
  - **Sentence:** ’сургуульд’
  - **Output:** `['сургуульд']`
  - **Analysis:** Fails to split the root from the case suffix, treating the entire inflected word as unique. This increases vocabulary size and worsens data sparsity.

- **Example 2:**
  - **Sentence:** ’номоороо’
  - **Output:** `['номороо']`
  - **Analysis:** Does not capture internal morphology, preventing the model from learning relationships between the base word and grammatical endings.

- **Example 3:**
  - **Sentence:** ’явчихлаа’
  - **Output:** `['явчихлаа']`
  - **Analysis:** Complex verb forms remain unsplit, making it harder for the model to generalize across similar conjugations.

Regex Tokenizer

- **Example 1:**
  - **Sentence:** ’А.Энхбат’
  - **Output:** `['А', '.', 'Энхбат']`
  - **Analysis:** Incorrectly splits a proper name, breaking semantic identity and potentially confusing entity recognition.

- **Example 2:**
  - **Sentence:** ’3.5км’
  - **Output:** `['3', '.', '5км']`
  - **Analysis:** Separating the decimal point disrupts the numeric value, leading to loss of quantitative meaning.

- **Example 3:**
  - **Sentence:** ’e-Mongolia’
  - **Output:** `['e', '-', 'Mongolia']`
  - **Analysis:** Hyphenated proper nouns are fragmented, reducing semantic coherence and harming contextual predictions.

BPE Tokenizer

- **Example 1:**
  - **Sentence:** ’байна’
  - **Output:** `['ба', 'йна']`
  - **Analysis:** Produces unnatural subwords that do not align with real morphemes, weakening semantic interpretability.
- **Example 2:**
  - **Sentence:** ’төрөлхийн’
  - **Output:** `['төр', 'өл', 'хийн']`
  - **Analysis:** Over-segmentation breaks meaningful units into arbitrary fragments, making it harder for the model to learn linguistic patterns.
- **Example 3:**
  - **Sentence:** ’төрсөн’
  - **Output:** `['төр', 'с', 'өн']`
  - **Analysis:** Splits the verb into excessively small pieces, increasing token count without adding useful information and potentially hurting fluency modeling.

Special Considerations for Mongolian:

- Mongolian uses Cyrillic script (different character properties than Latin)
- Agglutinative morphology (words formed by adding affixes)
- How does BPE handle morphological boundaries?
- Does regex tokenization work well with Cyrillic punctuation?

Summary: Which Tokenizer for Which Language? English:

- **Best tokenizer:** BPE
- **Reasoning:** Reduces vocabulary size while preserving meaningful subword patterns, leading to better generalization and improved handling of unseen words.

Mongolian:

- **Best tokenizer:** BPE
- **Reasoning:** BPE is particularly effective for Mongolian because it handles the agglutinative nature of the language by breaking down complex words into meaningful subword units. It reduces vocabulary size while preserving morphological information, which is crucial for a language with rich inflectional morphology and Cyrillic script characteristics.

3 Task 2: Language Modeling

3.1 Implementation Details (Task 2.1, 2.2)

3.1.1 Model Configuration

- **N-gram order:** 4 (4-grams)
- **Smoothing methods:** None (MLE baseline), Witten-Bell, Kneser-Ney
- **Kneser-Ney discount:** 0.75

3.1.2 Simplifying Assumptions

- **Greedy decoding:** Most likely token selected at each step (not beam search)
- **Sentence markers:** Padded with <s> (start) and </s> (end)
- **Generation termination:** Stops at </s>

3.2 Perplexity Results (Task 2.3)

Table 1: Perplexity Results

Tokenizer	No Smoothing	Witten-Bell	Kneser-Ney
Whitespace	[inf]	[89858.96]	[2240.72]
Regex	[inf]	[14371.2]	[549.39]
BPE	[inf]	[64.86]	[44.26]

Analysis of Perplexity Results: Best Overall Model:

- The combination of BPE tokenizer and Kneser–Ney smoothing gave the lowest perplexity.
- With a value of 44.26, it provided the most accurate and consistent predictions across all models.

Effect of Smoothing:

- **No smoothing:** All tokenizers resulted in infinite perplexity because unseen n-grams were assigned zero probability, making the model unable to generalize to new sequences.
- **Witten-Bell:** Significantly reduced perplexity compared to no smoothing by redistributing probability mass to unseen events, though performance varied depending on tokenization.
- **Kneser-Ney:** Delivered the best performance as it considers context diversity rather than relying only on frequency, leading to more accurate probability estimates.

Effect of Tokenization:

- **Whitespace:** Performed the worst due to very large vocabulary and severe sparsity, resulting in extremely high perplexity even with smoothing.
- **Regex:** Improved over whitespace by producing cleaner tokens and slightly reducing vocabulary size, but still suffered from sparsity inherent to word-level tokenization.
- **BPE:** Achieved dramatically lower perplexity because subword units reduce vocabulary size, improve coverage of rare words, and provide more reliable n-gram counts.

Key Observations:

- A clear pattern shows that reducing vocabulary size leads to lower perplexity by minimizing sparse contexts.
- There is a strong trade-off between vocabulary size and predictive confidence, with subword tokenization offering better statistical efficiency.
- Smoothing is critical in language modeling because it prevents zero probabilities and enables the model to handle unseen sequences effectively.

3.3 Qualitative Analysis (Task 2.4)

3.3.1 Auto Completions analysis

Whitespace + No Smoothing (3 examples)

- **Example 1:**
  - **Prefix :** ’3 . He’
  - **Model Output:** ’3 . He has only girl fans OBVIOUSLY . Only 1 or 2 boys around the world or even none like him . He is not as good a singer as many believe’
  - **Analysis:** The model generates a long but coherent continuation because the prefix likely exists in the training data. However, without smoothing it cannot handle unseen contexts, making the model brittle despite fluent output.

- **Example 2:**
  - **Prefix :** ’5 . He’
  - **Model Output:** ’5 . He doesn ’ t have my chair . Clearly discriminatory .’
  - **Analysis:** The sentence is grammatically correct but overly specific, suggesting memorization from training data. No smoothing limits generalization to new word sequences.

- **Example 3:**
  - **Prefix :** ’The reasons could’
  - **Model Output:** ’The reasons could be as many as you can just think about because when you are drawing others fan ’ s towards you then must definitely the competitive environment will be alarmingly high’
  - **Analysis:** Fluent but repetitive structure indicates reliance on frequent n-grams. The absence of smoothing increases the risk of zero probabilities for unseen phrases.

Whitespace + Witten-Bell (3 examples)

- **Example 1:**
  - **Prefix :** ’3 . He’
  - **Model Output:** ’3 . He has only girl fans OBVIOUSLY . Only 1 or 2 boys around the world or even none like him . He is not as good a singer as many believe’
  - **Analysis:** Output remains fluent while smoothing allows probability mass for unseen n-grams, improving robustness without significantly altering the learned structure.

- **Example 2:**
  - **Prefix :** ’5 . He’
  - **Model Output:** ’5 . He doesn ’ t have my chair . Clearly discriminatory .’
  - **Analysis:** The model produces a natural continuation with better generalization than no smoothing, demonstrating Witten-Bell’s ability to balance observed and unseen events.

- **Example 3:**
  - **Prefix :** ’The reasons could’
  - **Model Output:** ’The reasons could be as many as you can just think about because when you are drawing others fan ’ s towards you then must definitely the competitive environment will be alarmingly high’
  - **Analysis:** Sentence flow is preserved, and smoothing helps prevent probability collapse, though some verbosity remains due to common phrase patterns.

Whitespace + Kneser-Ney (3 examples)

- **Example 1:**
  - **Prefix :** ’3 . He’
  - **Model Output:** ’3 . He has no talent and his voice is like girl but he is a little boy growing now , when he is 18 / 20 years old , his voice grow’
  - **Analysis:** The continuation is more diverse, showing Kneser-Ney’s strength in modeling word continuation probabilities rather than raw frequency.

- **Example 2:**
  - **Prefix :** ’5 . He’
  - **Model Output:** ’5 . He doesn ’ t have a sponser . He should be by disney or something .’
  - **Analysis:** Produces a plausible but less memorized sentence, indicating improved handling of sparse contexts compared to simpler smoothing methods.

- **Example 3:**
  - **Prefix :** ’The reasons could’
  - **Model Output:** ’The reasons could be as many as you can just think about because when you are drawing others fan ’ s towards you then must definitely the competitive environment will be alarmingly high’
  - **Analysis:** Output is similar to other models because the prefix is frequent; however, Kneser-Ney generally improves predictions for rare continuations.

Regex + No Smoothing (3 examples)

- **Example 1:**
  - **Prefix :** ’3 . He’
  - **Model Output:** ’3 . he has completed more things already than most people do in their life time ! and its just the beginning’
  - **Analysis:** Cleaner token boundaries improve readability, but the lack of smoothing still restricts the model to seen sequences.

- **Example 2:**
  - **Prefix :** ’5 . he’
  - **Model Output:** ’5 . he doesn ’ t have my chair . clearly discriminatory .’
  - **Analysis:** Accurate continuation suggests strong training overlap, yet the model remains fragile for unseen patterns.

- **Example 3:**
  - **Prefix :** ’the reasons could’
  - **Model Output:** ’the reasons could be as many as you can just think about because when you are drawing others fan ’ s towards you then must definitely the competitive environment will be alarmingly high’
  - **Analysis:** Structured tokenization supports prediction, but zero-probability risks persist without smoothing.

Regex + Witten-Bell (3 examples)

- **Example 1:**
  - **Prefix :** ’3 . He’
  - **Model Output:** ’3 . he has completed more things already than most people do in their life time ! and its just the beginning’
  - **Analysis:** Combining structured tokens with smoothing improves reliability while maintaining grammatical coherence.

- **Example 2:**
  - **Prefix :** ’5 . he’
  - **Model Output:** ’5 . he doesn ’ t have my chair . clearly discriminatory .’
  - **Analysis:** The model generalizes better than the unsmoothed version, producing natural text even with moderately rare contexts.

- **Example 3:**
  - **Prefix :** ’the reasons could’
  - **Model Output:** ’the reasons could be as many as you can just think about because when you are drawing others fan ’ s towards you then must definitely the competitive environment will be alarmingly high’
  - **Analysis:** Smoothing prevents abrupt failures and supports stable sentence completion.

Regex + Kneser-Ney (3 examples)

- **Example 1:**
  - **Prefix :** ’3 . He’
  - **Model Output:** ’3 . he has no talent and his voice is like girl but he is a little boy growing now , when he is 18 / 20 years old , his voice grow’
  - **Analysis:** More varied wording reflects Kneser-Ney’s effective treatment of continuation probabilities alongside clean tokenization.

- **Example 2:**
  - **Prefix :** ’5 . he’
  - **Model Output:** ’5 . he doesn ’ t have a sponser . he should be by disney or something .’
  - **Analysis:** Generates a realistic continuation with improved handling of sparse data compared to simpler smoothing.

- **Example 3:**
  - **Prefix :** ’the reasons could’
  - **Model Output:** ’the reasons could be as many as you can just think about because when you are drawing others fan ’ s towards you then must definitely the competitive environment will be alarmingly high’
  - **Analysis:** Frequent prefix leads to similar output, but the method enhances robustness overall.

BPE + No Smoothing (3 examples)

- **Example 1:**
  - **Prefix :** `3 .</w> he</w>`
  - **Model Output:** `3 .</w> he</w> has</w> comple ted</w> more</w> things</w> already</w> than</w> most</w> people</w> do</w> in</w> their</w> life</w> time !</w> and</w> its</w> just</w> the</w> beg in ning</w>`
  - **Analysis:** Subword splits allow handling of rare words, but fragmented tokens reduce readability. No smoothing still limits unseen sequence prediction.

- **Example 2:**
  - **Prefix :** `5 .</w> he</w>`
  - **Model Output:** `5 .</w> he</w> does n't</w> have</w> a</w> sp on s er.</w> he</w> should</w> be</w> by</w> dis ne y</w> or</w> some thing .</w>`
  - **Analysis:** The model successfully constructs words from subunits, yet excessive segmentation harms linguistic clarity.

- **Example 3:**
  - **Prefix :** the</w> reasons</w> could</w>
  - **Model Output:** the</w> reasons</w> could</w> be</w> as</w> many</w> as</w> you</w> can</w> just</w> think</w> about</w> because</w> when</w> you</w> are</w> draw ing</w> others</w> f an's</w> t ow ards</w> you</w> then</w> must</w> de fin it ely</w> the</w> com
  - **Analysis:** Early truncation suggests probability issues, highlighting the importance of smoothing with subword models.

BPE + Witten-Bell (3 examples)

- **Example 1:**
  - **Prefix :** 3 .</w> he</w>
  - **Model Output:** 3 .</w> he</w> has</w> comple ted</w> more</w> things</w> already</w> than</w> most</w> people</w> do</w> in</w> their</w> life</w> time !</w> and</w> its</w> just</w> the</w> beg in ning</w>
  - **Analysis:** Smoothing stabilizes predictions while BPE manages rare vocabulary, though subword fragmentation remains noticeable.

- **Example 2:**
  - **Prefix :** 5 .</w> he</w>
  - **Model Output:** 5 .</w> he</w> does n't</w> have</w> a</w> sp on s er.</w> he</w> should</w> be</w> by</w> dis ne y</w> or</w> some thing .</w>
  - **Analysis:** Improved generalization is evident, but readability is slightly compromised due to token splits.

- **Example 3:**
  - **Prefix :** the</w> reasons</w> could</w>
  - **Model Output:** the</w> reasons</w> could</w> be</w. as</w> many</w> as</w> you</w> can</w> just</w> think</w> about</w> because</w> when</w> you</w> are</w> draw ing</w> others</w> f an's</w> t ow ards</w> you</w> then</w> must</w> de fin it ely </ w > the </ w > com
  - **Analysis:** The model continues reliably without probability collapse, demonstrating the benefit of combining smoothing with subword tokenization.

BPE + Kneser-Ney (3 examples)

- **Example 1:**
  - **Prefix :** 3 .</w> he</w>
  - **Model Output:** 3 .</w> he</w> has</w> only</w> girl</w> fans</w> ob vi ous l y.</w> only</w> 1</w> or</w> 2</w> boys</w> around</w> the</w> world</w> or</w> even</w> none</w> like</w> him.</w> he</w> is</w> un popular</w> and</w> people</w> are</w> his</w>
  - **Analysis:** Better continuation diversity shows Kneser-Ney working effectively with subwords, though text still appears segmented.

- **Example 2:**
  - **Prefix :** 5 .</w> he</w>
  - **Model Output:** 5 .</w> he</w> does n't</w> have</w> a</w> sp on s er.</w> he</w> should</w. be</w> by</w> dis ne y </ w > or </ w > some thing . </ w >
  - **Analysis:** Handles sparse contexts well and forms plausible completions despite aggressive token splitting.

- **Example 3:**
  - **Prefix :** `the</w> reasons</w> could</w>`
  - **Model Output:** `the</w> reasons</w> could</w> be</w> as</w> many</w> as</w> you</w. can</w> just</w> think</w> about</w> because</w> when</w> you</w> are</w> draw ing</w> others</w> f an's</w> t ow ards</w> you</w> then</w> must</w. de fin it ely </ w > the </ w > com`
  - **Analysis:** Stable output reflects strong smoothing, but excessive segmentation suggests that BPE may be less ideal for word-level fluency in n-gram models.

3.3.2 Effect of Smoothing on Generation Quality

No Smoothing:

- Without smoothing, unseen n-grams receive zero probability, making the model highly rigid and prone to repetition. Text generation often resembles memorized training data and may terminate when an unseen context appears. While it can produce grammatically correct sentences on small datasets, it generally fails to generalize as data grows.

Witten-Bell:

- Witten–Bell redistributes probability mass to unseen events based on the number of unique continuations for a context, enabling more flexible text generation. It reduces repetition and prevents dead ends but may sometimes allow unlikely word combinations. Overall, it provides a balance between determinism and creativity.

Kneser-Ney:

- Kneser–Ney improves generation quality by considering how widely a word appears across different contexts rather than relying only on frequency. This leads to more natural transitions, better fluency, and fewer unnatural phrases. It is widely regarded as the most effective smoothing method for higher-order n-gram models.

3.3.3 Effect of Tokenization on Generation Quality

Tokenization plays a crucial role in language modeling because it defines the unit on which probabilities are learned. The choice of tokenizer directly impacts vocabulary size, data sparsity, and the model's ability to generalize, all of which influence the fluency and coherence of generated text.

Whitespace:

- Splits text only on spaces, resulting in a very large vocabulary.
- Causes severe data sparsity, making higher-order n-gram predictions unreliable.
- Generated text is often repetitive, rigid, and closely resembles memorized training data.

Regex:

- Separates punctuation and symbols, producing cleaner and more consistent tokens.
- Slightly improves probability estimation but still suffers from vocabulary explosion.
- Text generation is somewhat more structured than whitespace but remains limited by rare words.

BPE:

- Breaks words into frequent subword units, greatly reducing vocabulary size.
- Minimizes sparsity and improves the model's ability to handle unseen words.
- Produces more fluent, flexible, and natural-sounding generated text.

4 Conclusion

Tokenization:

- For English **Byte Pair Encoding (BPE)** performed best because it reduced vocabulary size while preserving meaningful subword patterns, leading to better generalization and improved handling of unseen words.
- For Mongolian **BPE** was most effective due to the language's rich morphology, where many word forms are created through suffixes. Subword tokenization helped capture these patterns and significantly reduced sparsity.
- **Trade-offs**: Tokenization requires balancing vocabulary size, meaningful representation, and handling of unknown words. Word-level methods preserve full semantic units but create large vocabularies and struggle with unseen terms. In contrast, BPE reduces vocabulary size and handles unknown words effectively through subwords, though tokens may be slightly less interpretable.

Language Modeling:

- **Best smoothing method**: Kneser–Ney smoothing performed best because it considers the diversity of contexts in which words appear, producing more reliable probability estimates and more natural text generation.
- **Best tokenizer for LM**: BPE proved most suitable for language modeling as it minimizes sparsity, stabilizes probability distributions, and improves both perplexity and generation fluency.
- **Perplexity vs. generation quality**: Lower perplexity generally indicates better predictive performance, but it does not always guarantee superior text generation. Generation quality depends on how probability mass is distributed; overly confident models may produce repetitive text, while well-balanced models generate more natural and diverse language.

5 References

1. CS224N – Language Modelling and Smoothing
2. Bill McCartney – NLP Lunch Tutorial: Smoothing
3. Jurafsky & Martin Chapter 3 – N-gram Language Models
4. Jurafsky & Martin Appendix B – Kneser-Ney Smoothing
5. Wikipedia – Mongolian Language