# Natural Language Processing

Assignment 01 (Fall 2025)

Instructor: Ayesha Enayet

August 31, 2025

## Question 1: Train BPE vs. WordPiece From Scratch & Compare

### Learning Goals

- Understand how **Byte-Pair Encoding (BPE)** and **WordPiece** build subword vocabularies.
- Implement both tokenizers **from scratch** using only the Python standard library.
- Inspect the **first 10 merges** for each algorithm.
- Train both tokenizers on the q1_corpus_near1000 corpus available at `https://drive.google.com/drive/folders/1po9naPNNrRhZl5rzvMd66MHu3svTerld?usp=drive_link`.
- Compute token/subword **frequencies on the original corpus**, list the **top-10** for each tokenizer.
- Write a brief **comparative analysis** and **report challenges**.

### Rules & Constraints

- Use only the Python standard library (`re`, `collections`, etc.). **No** external tokenization libraries (e.g., `tokenizers`, `sentencepiece`, `transformers`).
- **Vocabulary cap** = **100** tokens for both methods (include single-character base tokens and any special markers).
- Print and clearly label:
  1. First **10 merges** for **BPE** and for **WordPiece**.
  2. **Top-10 tokens** (token, frequency) for each trained tokenizer on the **original corpus**.
- Follow the algorithmic variants discussed in class: BPE merges the most frequent *adjacent pair* (use an end-of-word marker, e.g., `</w>`); WordPiece uses greedy longest-match tokenization with a continuation prefix (e.g., ) and merges by your class scoring rule.

$$\mathrm{score}(x,y)=\frac{\mathrm{count}(x\triangleright y)}{\mathrm{count}(x)\,\mathrm{count}(y)}.$$

### Mandatory Implementation Blocks (Must Appear Clearly in Your Code)

Your implementation must include separate functions for the following operations. These may call your helper functions.

1. **Corpus Reader**: implement `read_text_file(...)`. You may call helper functions.
2. **Single-Step Merge Implementations (BPE & WordPiece)**: implement `bpe_merge_once(...)` and `wordpiece_merge_once(...)`; these must be explicitly invoked during training to print the first 10 merges. You may call helpers.
3. **Top-K Frequency Reporting**: implement `print_top_k(...)` to print the most frequent tokens/subwords for each trained tokenizer. You may call helpers.

### What to Submit

1. **Code**: a single `bpe_wordpiece_from_scratch.py` containing the three required blocks above, BPE/WordPiece training & tokenization, and clearly printed outputs.
2. **Output File** (copy into your report as well):
   - First 10 merges for BPE and WordPiece (clearly labeled).
   - Top-10 tokens (token, frequency) for each trained tokenizer on the original corpus.
3. **Short report (PDF, 1–2 pages)**:
   - Comparative analysis: early merges, morphological units, function words, affixes.
   - Challenges: tie-breaking, punctuation/Unicode, speed/memory, boundary effects, reproducibility.

### Grading Rubric (100 pts)

- Correct BPE implementation (25 pts).
- Correct WordPiece implementation (25 pts).
- Report (Comparative analysis) (35 pts).
- Code quality & documentation (15 pts).

**Academic Integrity:** This is an individual assignment. Cite any resources you consult. Do not use existing tokenization libraries or copy implementations.

## Question 2: Record Extraction Using Regex

Write a program that reads a text file containing 50 records in the form `{abc, 124, 23/45/67}` where each record contains exactly three fields: **name**, **phone**, and **date of birth (dob)**. The *order varies by record* (e.g., sometimes `name` first, sometimes `dob` first). Records are separated by curly braces (e.g., `}{`). Using **regular expressions**, extract the **name** and **dob** from each record and write a new file in which each line is `name,dob`.

## Input format

- The file is UTF-8 text. Records appear like `{...}{...}{...}` and may span lines.
- Input file Question2_input URL: https://drive.google.com/drive/folders/1po9naPNNrRhZl5rzvMd66MHu3s usp=drive_link
- Each record contains three comma-separated fields in *any order*:
  - **name**: alphabetic words (may include spaces, apostrophes, hyphens, dots).
  - **phone**: a number-like field (digits with optional separators, e.g., `+92-300-1234567`, `124`).
  - **dob**: a date in *varying formats*, e.g., `dd/mm/yyyy`, `d/m/yy`, `yyyy-mm-dd`, `dd-mm-yyyy`, `mm/dd/yyyy`, `12 Mar 1980`, `March 12, 1980`.

## Tasks

1. **Read the file** and **extract records** using a regex that finds balanced curly-brace blocks.
2. **Within each record**, use **regex-based classification** to identify which token is `name`, which is `dob`, and which is `phone`.
   - Use a *date* alternation pattern that covers multiple formats (see "Regex sketch" below).
   - Use a *phone* pattern that primarily matches digit-heavy strings (allow `+ - ( )` separators).
   - Treat the remaining token (that is not phone or date) as *name*; validate with a name regex.
3. **Normalize dates** to ISO `YYYY-MM-DD`. If the year is two digits, define a clear rule (e.g., `00-24` → 2000–2024; otherwise 1900–1999) and state it in a comment.
4. **Write output** to a new file (UTF-8): one record per line in the form `name,dob` (ISO). Preserve original name spacing/punctuation.

## Constraints

- You **must** use Python's `re` for field extraction/classification. No external parsing/tokenization libraries.
- You may use the Python standard library (e.g., `datetime`) for date normalization *after* regex extraction.
- Handle stray whitespace robustly (`\s*` around commas and inside braces). Ignore empty or malformed records gracefully.

## Required functions (clearly separated in your code)

a) `read_text(...)`

b) `extract_records(...)`

c) `classify_tokens(...)`

d) `normalize_dob(...)`

e) `write_pairs(...)`

### Deliverables

- Code (with short comments).
- Output file.
- Report.

## Report Submission (Regex Structuring Task)

### What to include

1. **Screenshot of Output File**
   Insert a clear screenshot of your final `name,dob` output file (one record per line).
2. **Regular Expressions Used (with brief descriptions)**
   List every regex you used to (i) extract records, (ii) split/classify tokens, and (iii) detect dates/-phones/names. Provide a short, plain-English purpose for each.
3. **Challenges Faced & Decisions Made**
   Briefly discuss:
   - Ambiguities between phone vs. date vs. name and how your regex resolved them.
   - Handling of whitespace, punctuation, two-digit years, and inconsistent date formats.
   - Any rejected approaches (and why) and how you validated correctness.

### Submission Format

- Submit a single PDF titled `Assign01_Report_YourName.pdf`.
- Include the screenshot (embedded), the regex table, and the challenges section.
- Keep it concise (1–2 pages). Ensure all text in screenshots is legible.

### Grading (100 pts)

- Correct regex-based record extraction and token classification (50 pts)
- Accurate date normalization and output formatting (10 pts)
- Code organization and clarity (10 pts)
- Report (30 pts)

**Post-Submission Demo (Required).** After the submission deadline and the official announcement, each student **must** visit my office during office hours to demonstrate their code. Failure to complete the demo within the announced window will result in a penalty of 75 pts.