Subword Tokenization Choices and Robustness: A Lightweight Evaluation on CoNLL-2003

Abstract

We evaluate common subword tokenization schemes such as Unigram, BPE, and WordPiece, on a sentence-level task derived from CoNLL-2003 (originally an NER dataset). To reuse a classic text-classification pipeline, we convert each sentence into (i) a binary label indicating whether the sentence *contains any named entity* or (ii) a multi-class label for the dominant entity type. We train tokenizers (target vocab ≈10k) then build TF-IDF features over tokenizer outputs and train a Logistic Regression classifier. We study robustness via small character noise (~5% random substitutions) and report accuracy on clean vs. noisy inputs.

Using binary “contains_entity” variant, Unigram achieves the highest clean and noisy accuracy, while BPE shows the smallest relative accuracy drop under noise. Absolute differences are modest (~0.5–1.0%).

Motivation and Problem Setup

Why this matters. Tokenization is the first and often most underrated step in an NLP stack. Different subword algorithms make different trade-offs around unknown words (OOV), vocabulary size, morphology, and robustness to typos. Rather than benchmarking inside a large neural model (which confounds many variables), we isolate tokenization by pairing it with a simple, transparent classifier (TF-IDF + Logistic Regression). This lets us:

- Inspect token distributions (Zipf behavior).
- Examine noises.
- Compare accuracy drops from clean to noisy text as a proxy for robustness.

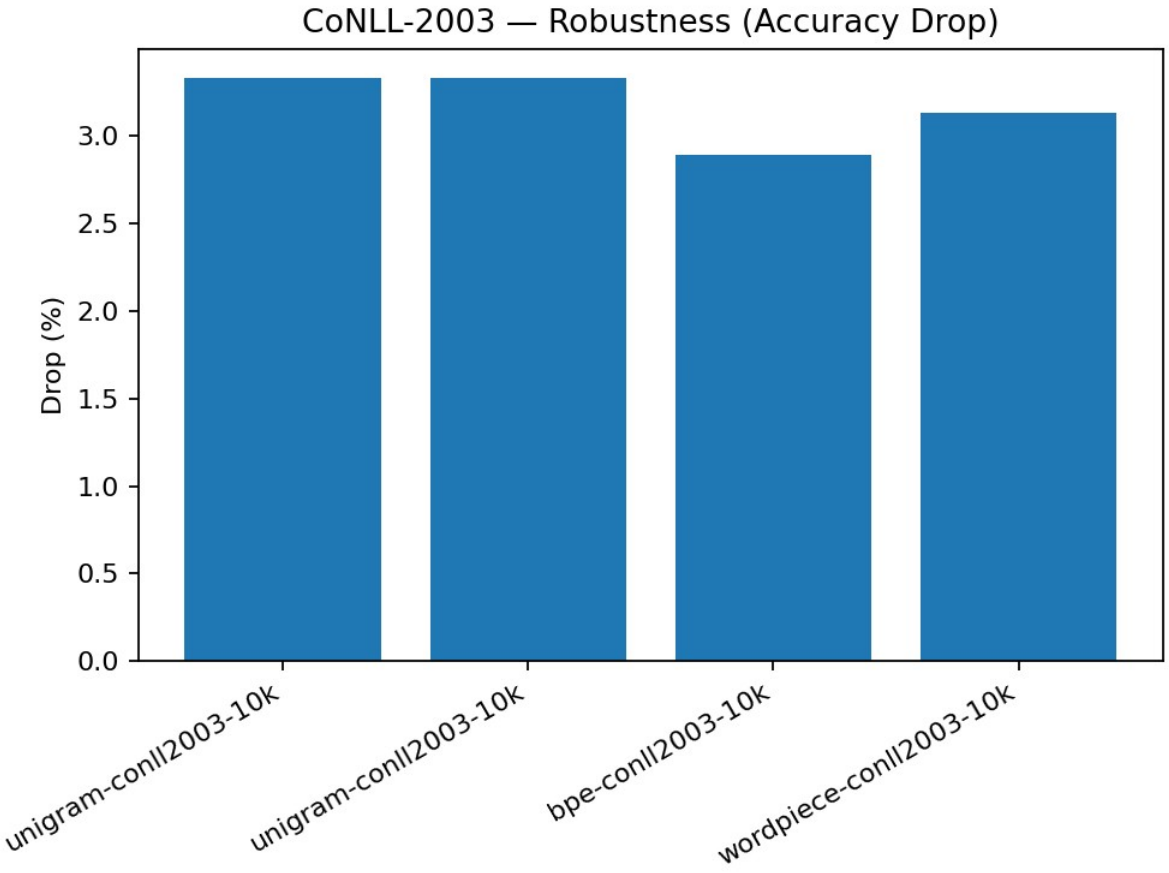
Task adaptation. CoNLL-2003 provides tokens and token-level NER tags. We convert each sentence to text and derive one of two sentence-level labels:

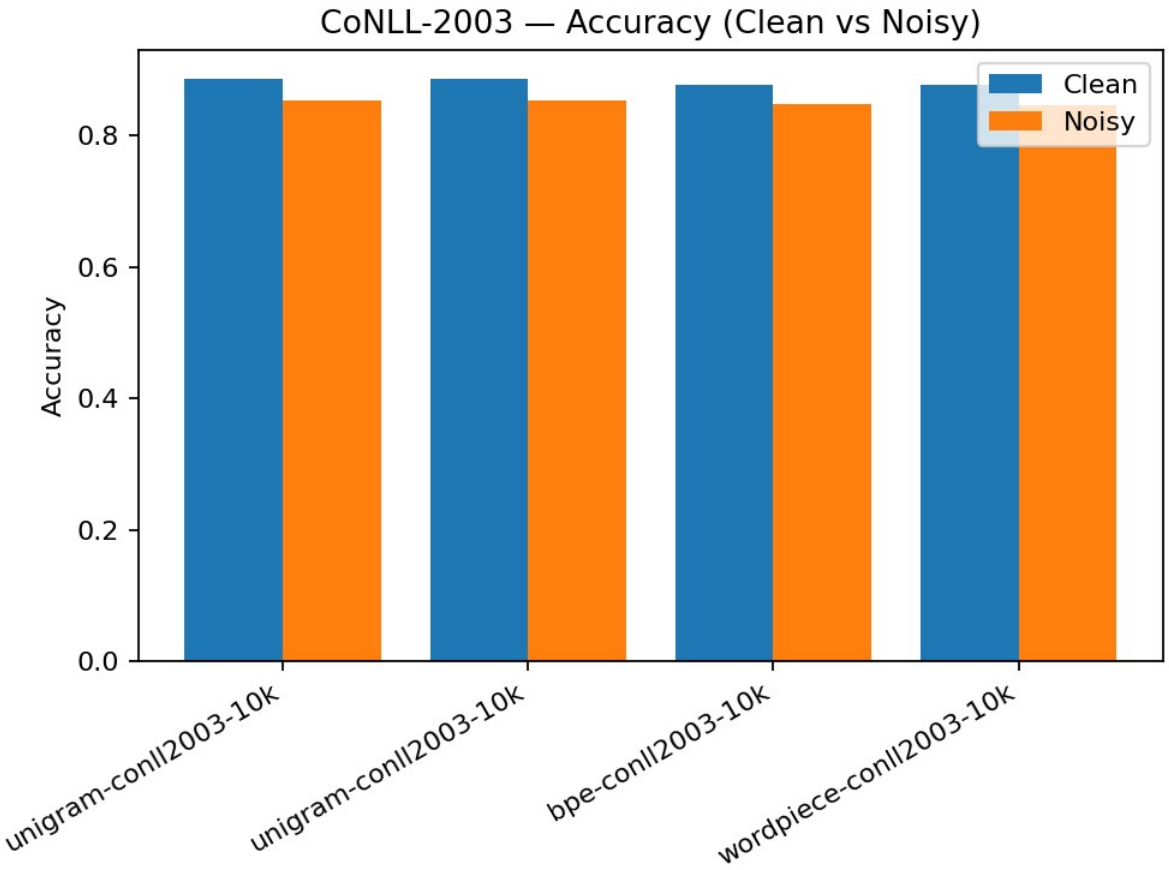
- contains_entity (binary): 1 if any token has a non-O NER tag.
- major_entity (multi-class): most frequent entity type in the sentence among {PER, ORG, LOC, MISC}; NONE if no entities.

This reframing allows standard text classification while still probing tokenization on entity-heavy text.

Data and Preprocessing

- Dataset. CoNLL-2003 (English) via the dataset's Parquet branch (no remote loader scripts). The prep cell writes:
    - corpus/conll2003_corpus.txt — one sentence per line
    - data/conll2003_contains_entity.csv — columns: text, label
- Normalization. Tokenizer training uses NFD → Lowercase → StripAccents and Whitespace pre-tokenization
- Split. We perform a fresh 80/20 stratified train/test split inside the analysis step.





Models and Algorithms

Tokenizers (HF tokenizers)

- Unigram (Kudo, 2018): Treats the vocabulary as a probabilistic model and prunes tokens via likelihood. Often yields flexible segmentations; with a modest vocab it can gracefully back off to shorter pieces.
- BPE (Sennrich et al., 2016): Greedily merges frequent character sequences. Typically produces longer frequent chunks, fewer tokens per word, and very low OOV.
- WordPiece (Schuster & Nakajima, 2012): Similar intuition to BPE but optimizes an explicit likelihood; ubiquitous in BERT-style models.

Training settings.
Target vocab ≈10k; specials ["[UNK]","[CLS]","[SEP]","[PAD]","[MASK]"]; Whitespace pre-tokenizer; NFD → lowercase → strip accents normalizer.

Unknown Words (OOV)

Subword models largely eliminate OOV by fragmenting unseen words into smaller known pieces. We still include [UNK] for safety (e.g., weird Unicode), and TF-IDF treats [UNK] like any other token.

Features and Classifier

- Pre-tokenized TF-IDF. We encode each sentence to tokens, join with spaces, and feed to TfidfVectorizer with a fixed vocabulary (the tokenizer's tokens). This guarantees a clean 1:1 mapping (token ↔ feature).
- Classifier. Logistic Regression (max_iter=1000), chosen for stability and interpretability.

Evaluation Methodology

- Noise model. ~5% character substitutions on alphabetic characters in the test set (keyboard-agnostic), then re-tokenize the perturbed text.
- Metrics. Accuracy on clean and noisy test sets; Drop (%) = (clean − noisy) × 100. We also emit a full classification report (precision/recall/F1 by class).
- Artifacts.
    - Text report: results/classification_report_<tokenizer>.txt
    - Scoreboard: results/full_results.csv
    - Figures: Zipf + final charts in results/

Results

Tokenizer	Vocab	Clean Acc	Noisy Acc	Drop (%)
unigram-conll2003-10k	9,374	0.8865	0.8532	3.33
bpe-conll2003-10k	10,000	0.8768	0.8479	2.89
wordpiece-conll2003-10k	10,000	0.8768	0.8455	3.13

Key takeaways:

- Best clean accuracy: Unigram at 88.65%, about +0.97 pp over BPE/WordPiece (87.68%).
- Best noisy accuracy (absolute): Unigram at 85.32%, about +0.53 pp over BPE (84.79%) and +0.77 pp over WordPiece (84.55%).
- Best robustness (smallest relative drop): BPE, 2.89% vs WordPiece 3.13% vs Unigram 3.33%.

Interpretation.

- Unigram's probabilistic vocabulary pruning yielded a slightly smaller effective vocab (9,374) than the 10k target (typical behavior when low-utility tokens are dropped). That smaller, denser vocab likely contributed to strong clean accuracy and the best absolute noisy accuracy.
- BPE's lowest drop (%) suggests that, relative to its own clean baseline, it degrades a bit less under noise, even if its absolute noisy accuracy trails Unigram by ~0.5 pp.
- WordPiece and BPE tie on clean accuracy here; WordPiece's drop is a hair larger than BPE's.

Caveats.

- Differences are small (~0.5–1.0%); with another random split or seed we might see slight reshuffling. A simple 5-fold CV would quantify the variance.
- Lowercasing removes case cues that often matter for entities; retaining case (remove Lowercase() in training) may help across all tokenizers, possibly changing the ranking.

Conclusions

We hold the downstream classifier fixed and swap only the tokenizer. That isolates differences in granularity, vocabulary composition, and fallback behavior under character noise.

- For best raw accuracy on both clean and slightly corrupted text, Unigram has a small edge.
- For relative robustness (minimizing % drop from clean to noisy), BPE looks marginally better.
- WordPiece behaves similarly to BPE on clean data with slightly more drop under noise in this setup.

Bottom line. For this sentence-level "contains any entity" prediction on CoNLL-2003 text, the differences are modest.

Future Work

Below are potential future items that we could undertake to expand on the work in this project:

1. **Vocab size sweeps.** Try 5k / 10k / 30k. Expect larger vocabs to lift clean accuracy but possibly reduce noise robustness as merges get brittle.
2. **Case sensitivity.** Remove Lowercase() and retrain to preserve capitalization—likely beneficial for entity-rich text.
3. **Richer noise models.** Keyboard-adjacent flips, deletions, insertions; word-level shuffles; even synthetic OCR noise.
4. **Byte-level BPE baseline.** Add a GPT-2-style tokenizer to test an extreme OOV-free setup; compare feature size and robustness.
5. **Subword regularization.** For Unigram, sample multiple segmentations at train time—often improves robustness.

6. **Beyond linear models.** Keep tokenizer fixed but swap the classifier (small CNN/Transformer) to separate tokenizer vs. capacity effects.
7. **Error analysis.** Inspect LR coefficients and misclassifications to see which subwords drive false positives/negatives; break out per-sentence length and per-entity-type slices.