Natural Language Processing Course

Week 8: Tokenization and Subword Models

Joerg R. Osterrieder
www.joergosterrieder.com

Week 8

Tokenization

The Hidden Foundation of Every LLM

The Word That Broke Google Translate

2016: A user typed "Pneumonoultramicroscopicsilicovolcanoconiosis"

Result: System crashed.[1]

The problem:

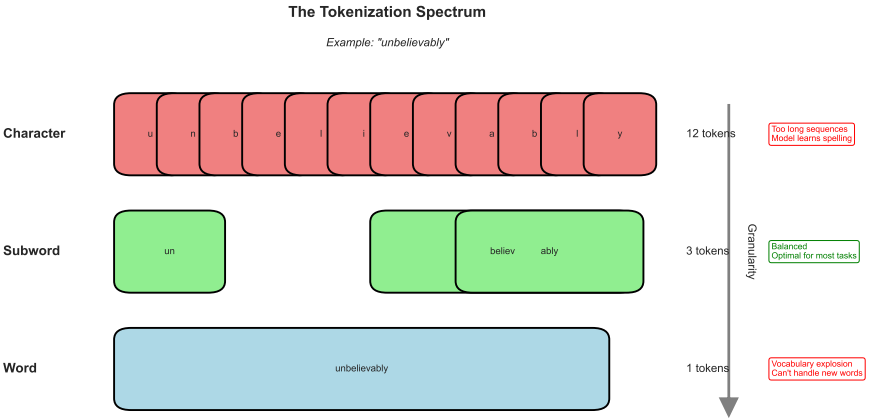
- English: 170,000 words in current use
- Medical terms: 100,000+ additional
- New words daily: "COVID-19", "cryptocurrency", "mansplaining"
- Misspellings: "recieve", "definately", "occured"
- Other languages: 7,000+ languages worldwide!

Can't have a token for every possible word - memory explosion!

The dilemma: How do you handle infinite vocabulary with finite memory?

[1] Simplified example - real systems had various OOV issues

The Tokenization Spectrum: Characters vs Words



The Goldilocks problem:

Tokenization Powers Every Modern LLM (2024)

Model Vocabularies:

- GPT-2: 50,257 tokens
- GPT-3/4: 50,257 tokens
- BERT: 30,522 tokens
- T5: 32,000 tokens
- LLaMA: 32,000 tokens

Why These Numbers?

- Power of 2 for efficiency
- Covers 99.9% of text
- Balances sequence length
- Works across languages

Tokenization Methods:

- BPE: GPT family[2]
- WordPiece: BERT family
- SentencePiece: T5, mT5
- Unigram: Some Japanese models

Critical Impact:

- Wrong tokenization $\rightarrow$ 50% performance drop
- Affects prompt cost (GPT-4: \$0.01/1K tokens)
- Determines max context length
- Controls multilingual ability

Tokenization is the most important choice you've never heard of

[2] Byte-level BPE specifically for GPT-2 onwards

Week 8: What You'll Master

By the end of this week, you will:

- **Understand** why subword tokenization dominates
- **Implement** Byte Pair Encoding from scratch
- **Master** the trade-offs in vocabulary design
- **Analyze** tokenization's impact on different languages
- **Build** your own tokenizer for any domain

Core Insight: Break words into learnable pieces

The Two Extremes: Characters vs Words

Example sentence: "The quick brown fox jumps"

Character tokenization:

- Tokens: [T, h, e, _, q, u, i, c, k, _, b, r, o, w, n, _, f, o, x, _, j, u, m, p, s]
- Length: 25 tokens
- Vocabulary: 100 (all ASCII)
- ✓ Handles any text
- ✗ Sequences too long
- ✗ Model must learn spelling

Word tokenization:

- Tokens: [The, quick, brown, fox, jumps]
- Length: 5 tokens
- Vocabulary: 170,000+ (English)
- ✓ Efficient sequences
- ✗ Out-of-vocabulary words
- ✗ Can't handle typos

Neither extreme works well - we need a middle ground

Byte Pair Encoding: Learning to Compress

The brilliant idea (1994):[3] Compress text by merging frequent pairs

Example: "low lower lowest"

1. Start with characters: l o w _ l o w e r _ l o w e s t
2. Count pairs: (l,o)=3, (o,w)=3, (w,e)=2, (e,r)=1...
3. Merge most frequent: "lo"
4. New: lo w _ lo w e r _ lo w e s t
5. Repeat: merge "low"
6. Final: low _ low er _ low est

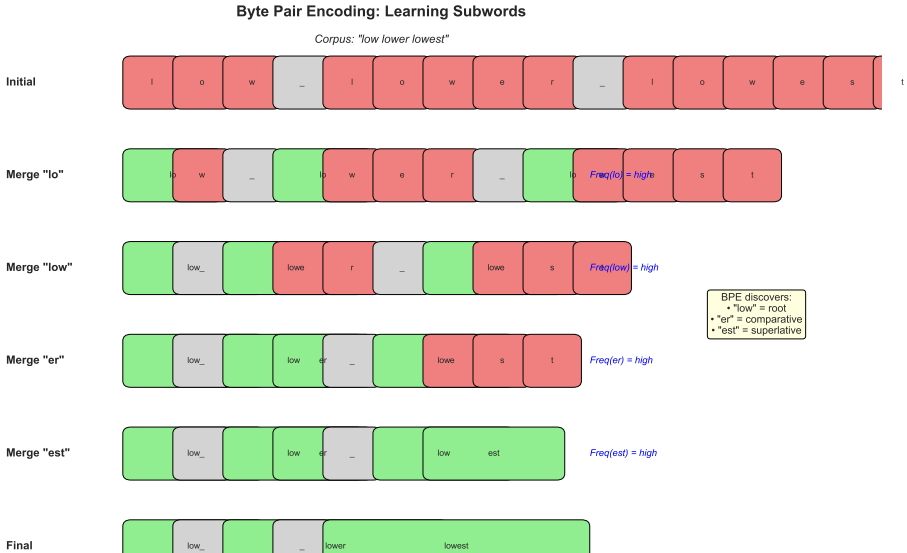
Result: Learned meaningful subwords!

- "low" = common root
- "er" = comparative suffix
- "est" = superlative suffix

BPE discovers linguistic structure automatically!

[3] Gage (1994) for compression; Sennrich et al. (2016) for NMT

BPE in Action: Building a Vocabulary



Implementing BPE: The Core Algorithm

```python
import re
from collections import defaultdict, Counter

class BytePairEncoding:
    def __init__(self, vocab_size):
        """Initialize BPE tokenizer"""
        self.vocab_size = vocab_size
        self.word_tokenizer = re.compile(r'\w+|[^\w\s]')
        self.vocab = {}
        self.merges = []

    def get_stats(self, words):
        """Count frequency of adjacent pairs"""
        pairs = defaultdict(int)
        for word, freq in words.items():
            symbols = word.split()
            for i in range(len(symbols) - 1):
                pairs[symbols[i], symbols[i + 1]] += freq
        return pairs

    def merge_vocab(self, pair, words):
        """Merge most frequent pair"""
        bigram = ' '.join(pair)
        replacement = ''.join(pair)
        new_words = {}
        for word in words:
            new_word = word.replace(bigram, replacement)
            new_words[new_word] = words[word]
        return new_words

    def train(self, text, num_merges):
        """Train BPE on text corpus"""
        # Split into words
        words = self.word_tokenizer.findall(text.lower())

```

Algorithm Steps:

- Start with character vocabulary
- Count all adjacent pairs
- Merge most frequent pair
- Repeat until vocab size reached

Design Choices:

- End-of-word token ¡/w¿
- Preserves word boundaries
- Case normalization optional
- Typically 10K-50K merges

Complexity:

- Training: O(NM) for N words, M merges
- Encoding: $O(L^2)$ for length L
- Memory: O(V) for vocabulary V

Tokenizer Comparison: Real Examples

Text: "Tokenization is surprisingly important for performance"

GPT-2 (BPE): `["Token", "ization", " is", " surprisingly", " important", " for", " performance"]`
7 tokens

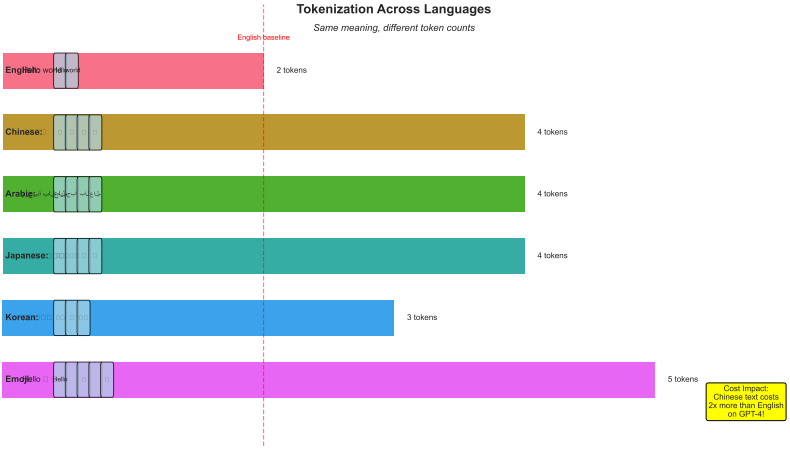
BERT (WordPiece): `["token", "##ization", "is", "surprisingly", "important", "for", "performance"]`
7 tokens (note: ## indicates continuation)

Character-level: `["T", "o", "k", "e", "n", "i", "z", "a", "t", "i", "o", "n", " ", ...]`
47 tokens!

Impact on rare words:
"Pneumonoultramicroscopicsilicovolcanoconiosis" $\rightarrow$

- BPE: ["P", "neum", "ono", "ult", "ram", "ic", "ros", "cop", "ic", "sil", "ico", "vol", "can", "oc", "on", "ios", "is"]
- Still handles it! (17 subwords vs 45 characters)

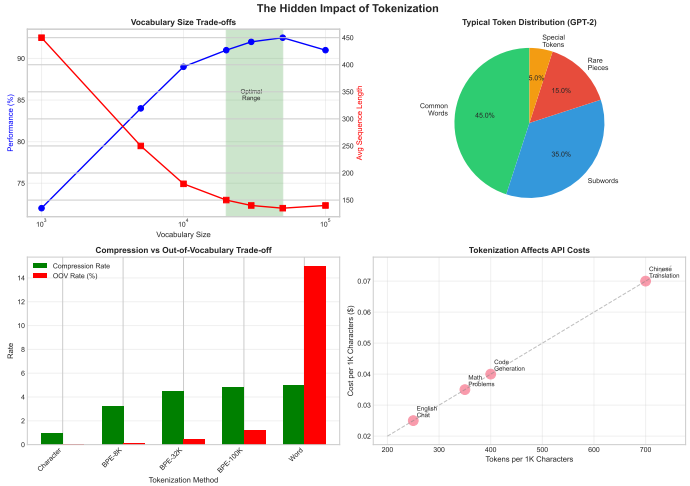
Tokenization Across Languages



Key challenges:

- Chinese/Japanese: No spaces between words
- Arabic/Hebrew: Right-to-left scripts
- Korean: Syllable blocks (hangul)

Tokenization's Hidden Impact



Key Insights

- Wrong tokenizer: 50% performance drop

State-of-the-Art Tokenizers (2024)

Byte-level BPE (GPT family):

- Handles ANY text (even emojis)
- No unknown tokens
- Consistent across languages
- Used by: GPT-2/3/4, RoBERTa

SentencePiece (T5, mT5):

- Language independent
- Reversible tokenization
- BPE or Unigram options
- Handles 100+ languages

Recent Innovations:

- **CANINE:** Character-level (2021)
- **ByT5:** Byte-level tokens
- **PIXEL:** Image-based text
- **Charformer:** Learnable tokenization

Tokenizer Selection:

- English only: BPE (30-50K)
- Multilingual: SentencePiece
- Code: Include special tokens
- Domain-specific: Custom training

2024 trend: Larger vocabularies (100K+) for better multilingual support

Tokenization Best Practices

1. **Vocabulary Size Trade-offs:**
   - Small (8K): Long sequences, better generalization
   - Medium (32K): Standard choice, balanced
   - Large (100K+): Shorter sequences, more memory

2. **Special Tokens:**

PAD , [UNK], [CLS], [SEP], [MASK]

   - Domain tokens: [CODE], [MATH], [URL]
   - Control tokens: [INST], [/INST]

3. **Common Pitfalls:**
   - Tokenizer/model mismatch (50% accuracy drop!)
   - Forgetting to handle special characters
   - Not preserving whitespace information
   - Case sensitivity mismatches

Real Example - GPT costs:

- "Hello world" = 2 tokens = $0.00002
- "Chinese text" = 4 tokens = $0.00004 (2x cost!)
- Emoji = 1-3 tokens depending on tokenizer

Week 8 Exercise: Build Your Domain Tokenizer

Your Mission: Create optimal tokenizer for your domain

Part 1: Analyze Tokenization Impact

- Compare GPT-2, BERT, T5 tokenizers
- Tokenize: English, code, multilingual text
- Measure compression rates
- Calculate cost differences

Part 2: Train Custom BPE

- Choose domain: medical, legal, code, etc.
- Collect domain corpus (10MB+)
- Train BPE with different vocab sizes
- Compare with general tokenizers

Part 3: Optimization Experiments

- Test character vs subword vs word
- Measure model performance impact
- Analyze out-of-vocabulary rates
- Optimize for your use case

You'll discover: Why tokenization is make-or-break for LLMs!

Key Takeaways: The Foundation of Language Models

What we learned:

- Tokenization solves the infinite vocabulary problem
- Subwords balance efficiency and coverage
- BPE learns meaningful units automatically
- Different tokenizers for different needs
- Critical impact on cost and performance

The evolution:

Characters → Words → Subwords → Learned tokenization

Why it matters:

- Determines model capabilities
- Affects all downstream tasks
- Can't fix after training!

Next week: Decoding Strategies
How do we generate coherent text from token probabilities?

References and Further Reading

Foundational Papers:

- Sennrich et al. (2016). "Neural Machine Translation of Rare Words with Subword Units"
- Kudo & Richardson (2018). "SentencePiece: Language-independent subword tokenizer"
- Schuster & Nakajima (2012). "Japanese and Korean voice search" (WordPiece)

Recent Advances:

- Clark et al. (2021). "CANINE: Character-level transformers"
- Xue et al. (2021). "ByT5: Token-free transformers"
- Tay et al. (2021). "Charformer: Fast character transformers"

Practical Resources:

- Hugging Face Tokenizers library
- Google SentencePiece
- OpenAI tiktoken library