# Natural Language Processing

## Week 8: Breaking Words into Pieces
## The Tokenization Revolution

NLP Course 2025

# Learning Objectives

**By the end of this lecture, you will understand:**

1. Why simple word-level tokenization fails (OOV problem)
2. How subword tokenization preserves meaning for rare words
3. The BPE algorithm and why it learns from frequency
4. Why WordPiece and SentencePiece improve on BPE
5. How tokenization impacts multilingual models

**Prerequisite**

**From previous weeks:**

- Word embeddings represent words as vectors (Week 2)
- Transformers process sequences of tokens (Week 5)
- Vocabulary size affects model size and training

# Table of Contents

# Act 1: The Out-of-Vocabulary Crisis

**Imagine you're building a translator...**

You train on common English words: cat, dog, run, happy, etc.

**Then a user types:**

"I feel **unhappiness** about this situation"

**Your model's vocabulary:**

**Known words:**

- happy
- happiness
- sad
- unhappy

**Unknown word:**

- unhappiness → `<UNK>`
- All meaning lost!
- Model has no idea what user meant

**The Dilemma:** Can't memorize every possible word. English has millions!

# Why Word-Level Tokenization Fails

**Real-World Application**

**Real scenario from GPT-2 (2019):**
Vocabulary size: 50,257 words
Reddit training data: Millions of rare words
Problem: 5-15% of test words were out-of-vocabulary!

**Three approaches to consider:**

**1. Character-Level**

- + Never OOV
- - Too many tokens
- - "the" = 3 tokens

**2. Word-Level**

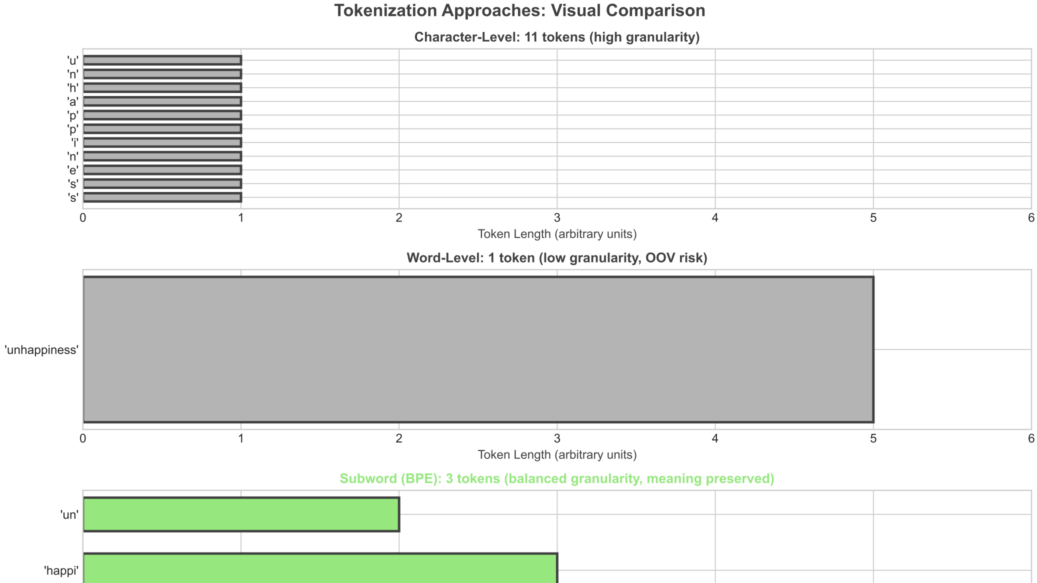
- + Natural units
- - High OOV rate
- - Huge vocabulary

**3. Subword-Level?**

- + Best of both?
- + Balanced tokens
- + Rare words OK

**Key Question:** How do we automatically find meaningful subword units?

# Visual: The Tokenization Spectrum



# Act 2: The BPE Solution

**Core Insight:** Learn subword units from frequency patterns

**The Algorithm (simplified):**

1. Start with character vocabulary: {a, b, c, ..., z}
2. Count all adjacent character pairs in corpus
3. Merge the most frequent pair into a new token
4. Repeat until vocabulary reaches target size

**Common Misconception**

**"BPE needs linguistic knowledge to find morphemes"**
FALSE! BPE is purely statistical. It discovers that "un-", "-ing", "-ness" are common patterns automatically, without knowing they're prefixes/suffixes.

**Key Properties:**

- Language-agnostic (works for any language)
- Data-driven (learns from your corpus)
- Vocabulary size is a hyperparameter (typically 30K-50K)

# BPE Example: Step-by-Step

**Let's walk through a tiny example...**

**Training corpus:**

```
low low low
lower lower
newest newest newest newest
```

**Initial representation (character + word boundary):**

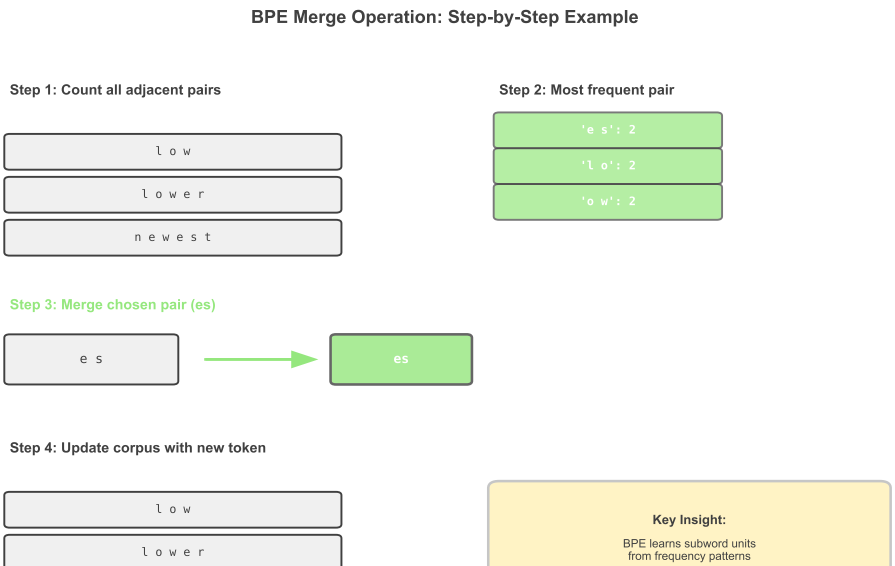
```
l o w </w>  (appears 3 times)
l o w e r </w>  (appears 2 times)
n e w e s t </w>  (appears 4 times)
```

**Count all pairs:**

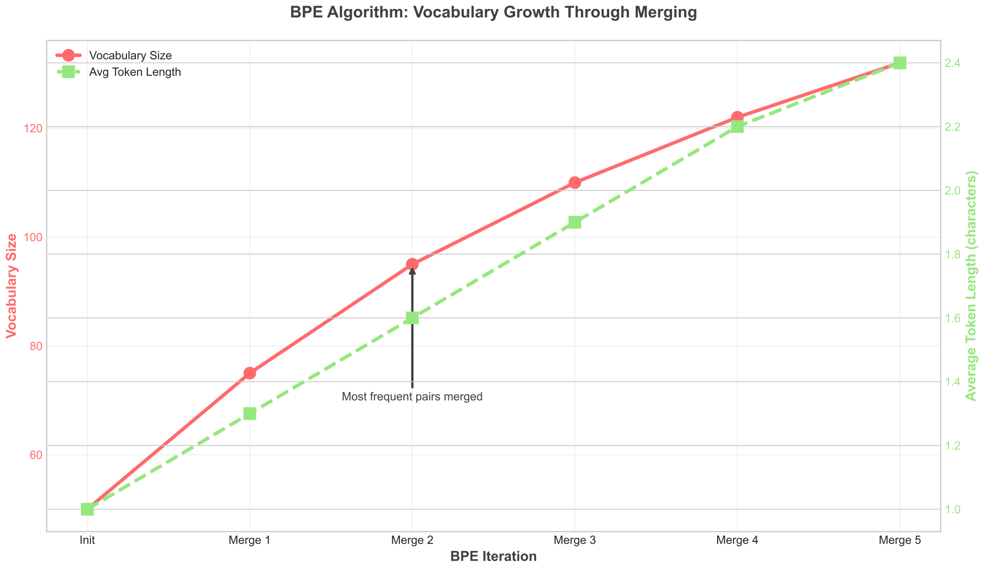
| Pair | Frequency |
|---|---|
| (e, s) | 4 |
| (l, o) | 5 |
| (o, w) | 5 |
| (n, e) | 4 |

**Merge most frequent:** (l, o) $\rightarrow$ lo OR (o, w) $\rightarrow$ ow

# Visual: BPE Merge Detail



# Visual: BPE Algorithm Progression



# Checkpoint 1: Understanding BPE

## Checkpoint

**Test your understanding:**

**Question 1:** What does BPE merge at each iteration?

A) Random character pairs
B) Linguistically meaningful morphemes
C) Most frequent adjacent pairs
D) Longest possible substrings

**Answer:** C - Most frequent adjacent pairs
BPE is purely frequency-based. It doesn't know about linguistics!

**Question 2:** Why does BPE never produce out-of-vocabulary tokens?

A) It memorizes all possible words
B) It can always fall back to character-level
C) It uses a special UNK token

**Answer:** B - Falls back to character-level
Since characters are always in vocabulary, any word can be decomposed.

# Act 3: The Rare Word Advantage

**Remember our "unhappiness" problem?**

Let's see how BPE handles it...

**After BPE training, vocabulary contains:**

- "un" (common prefix in: unable, unhappy, unknown, etc.)
- "happi" (appears in: happy, happiness, unhappy, etc.)
- "ness" (common suffix in: happiness, sadness, kindness, etc.)

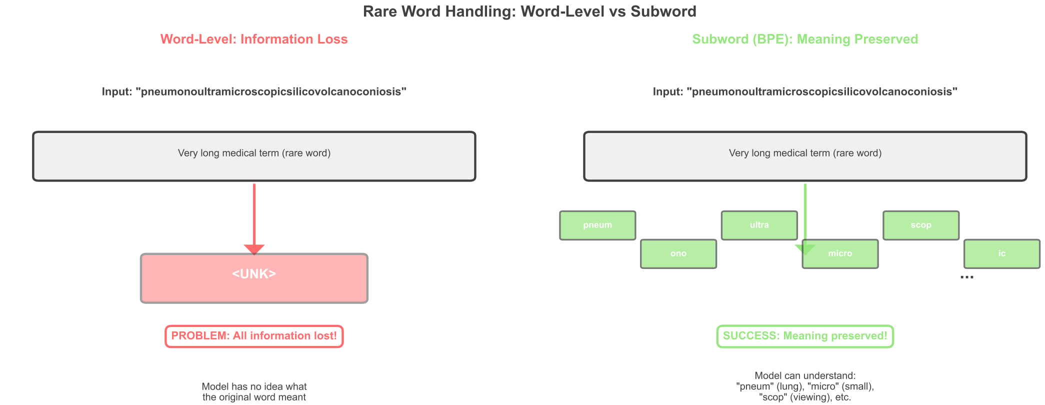
**At test time:**

$$\text{“unhappiness”} \rightarrow \text{[un] [happi] [ness]}$$

**Model can now understand:**

- "un" = negation prefix
- "happi" = emotional state (positive)
- "ness" = quality/state suffix
- Combined: negated positive emotional state = negative feeling

# Visual: Rare Word Handling Comparison




## Real-World Application

**Medical domain example:**

Word: "pneumonoultramicroscopicsilicovolcanoconiosis" (lung disease)

# The Vocabulary Size Trade-off

**How large should our vocabulary be?**

**Small vocabulary (5K-10K):**

+ Fast training
+ Memory efficient
- Many subword splits
- Longer sequences

**Large vocabulary (50K-100K):**

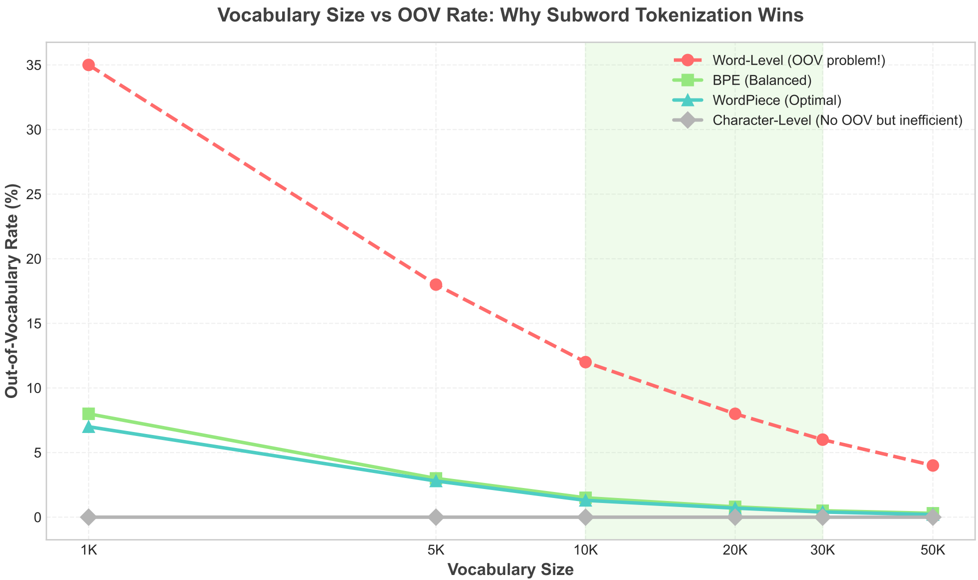
+ Fewer splits
+ More "whole words"
- Slower training
- More parameters

**Industry standard:** 30K-50K tokens (sweet spot)

**Intuition**

Think of it like image compression: Higher compression ratio = faster but lower quality. Lower compression = slower but better quality. Subword tokenization finds the optimal balance.

# Visual: Vocabulary Size vs OOV Rate



# Act 4: Improvements on BPE

**BPE was just the beginning...**

**1. WordPiece (Google, 2016):**

- Used in BERT, T5, many Google models
- **Key difference:** Merges based on likelihood increase (not just frequency)
- Formula: Choose merge that maximizes $P(\text{corpus})$
- **Result:** Slightly better language model perplexity

**2. SentencePiece (Google, 2018):**

- Works on raw text (no pre-tokenization needed)
- Language-agnostic (handles Chinese, Japanese, Arabic, etc.)
- Treats spaces as tokens (e.g., "␣hello")
- Used in: XLNet, ALBERT, T5, many multilingual models

**Common Misconception**

**"BPE is outdated, everyone uses WordPiece now"**
FALSE! GPT-2, GPT-3, GPT-4, RoBERTa, and many others still use BPE. Choice depends on specific use case.

# Comparison: BPE vs WordPiece vs SentencePiece

| Property | BPE | WordPiece | SentencePiece |
|---|---|---|---|
| Merge criterion | Frequency | Likelihood | Both supported |
| Pre-tokenization | Required | Required | Not required |
| Space handling | Special char | Special char | Treated as token |
| Multilingual | Good | Good | Excellent |
| Used in | GPT-2/3/4 | BERT | XLNet, T5 |

## Real-World Application

**Why does GPT-4 use BPE while BERT uses WordPiece?**
BPE: Simpler, faster training, good for autoregressive models
WordPiece: Better perplexity, good for masked language modeling
Both work well - choice is often historical/engineering preference

# The Multilingual Challenge

**Different languages have different characteristics...**

**English:**

- Space-separated words
- Moderate morphology
- "running" = run + ing

**German:**

- Compound words
- Rich morphology
- "Donaudampfschifffahrtsgesellschaftskapitan"

**Chinese:**

- No spaces between words
- Character-based writing
- Each character has meaning

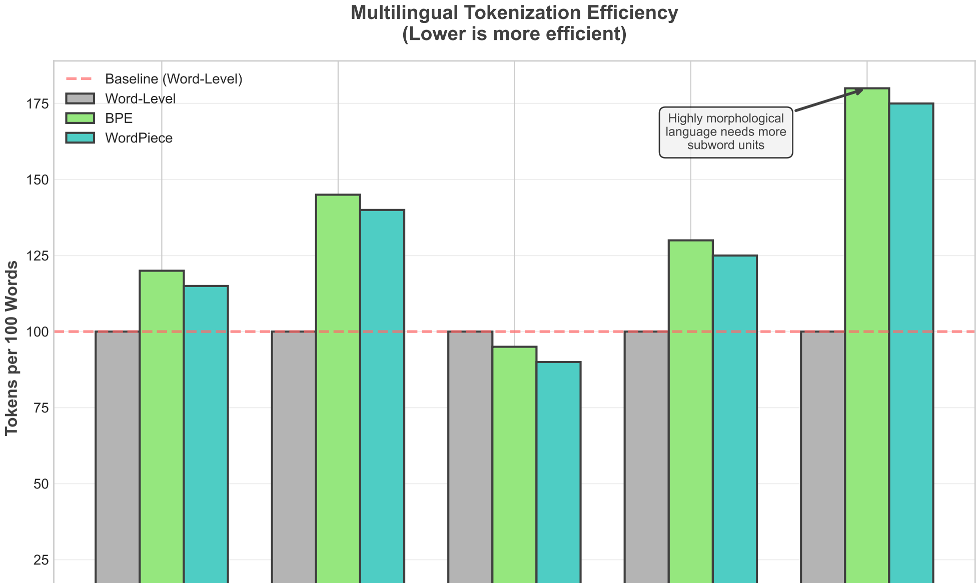
**Finnish:**

- Extremely complex morphology
- 15 grammatical cases
- "talossanikinko" = "in my house too?"

**Challenge:** How can one tokenization scheme work for all languages?

**Answer:** Subword tokenization adapts automatically! Languages with complex morphology get more subword splits, languages with simpler structure get fewer splits.

# Visual: Multilingual Tokenization Efficiency



## Checkpoint

**Test your understanding:**

**Question 1:** Why does Finnish need more BPE tokens than English for the same text?

A) Finnish words are longer

B) Finnish has complex morphology requiring more subword units

C) BPE doesn't work well for Finnish

D) Finnish has a larger alphabet

**Answer:** B - Complex morphology
Finnish words encode grammatical information through suffixes, so need more subword splits. This is a feature, not a bug - the model learns morphological patterns!

**Question 2:** What's the main advantage of SentencePiece over BPE for Chinese?

A) It understands Chinese grammar

B) No pre-tokenization needed (no spaces in Chinese)

C) It produces shorter sequences

**Answer:** B - No pre-tokenization needed
Chinese doesn't use spaces between words, so SentencePiece's raw text approach is ideal.

# Does Tokenization Really Matter?

**Yes! Tokenization significantly impacts model performance...**

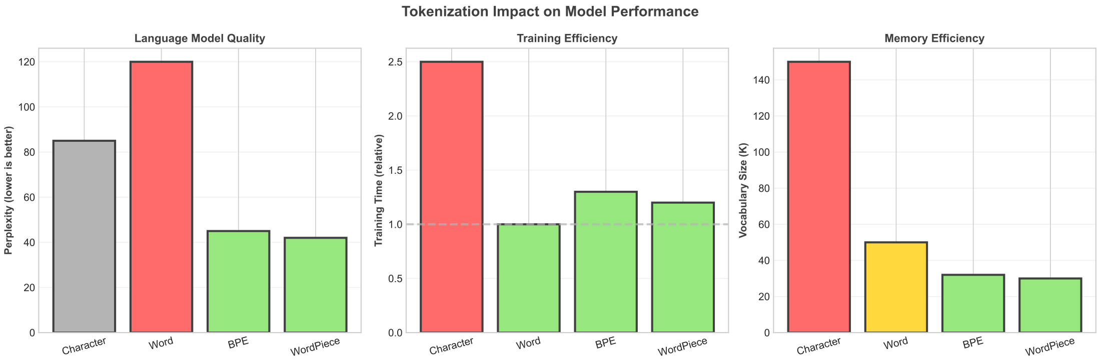
**Real-World Application**

**Case study: GPT-2 (2019)**
OpenAI experimented with different tokenization schemes:
Character-level: Perplexity = 85, Training time = 2.5x
Word-level: Perplexity = 120 (OOV problems!)
BPE (50K vocab): Perplexity = 45, Training time = 1.3x
Result: BPE became standard for GPT series

**Three key metrics affected:**

1. **Model quality:** Better tokenization = lower perplexity
2. **Training efficiency:** Balanced sequence length = faster training
3. **Memory usage:** Smaller vocabulary = fewer parameters

# Visual: Tokenization Impact on Performance




**Key Takeaway:** WordPiece and BPE achieve best balance across all three metrics

# Implementing BPE in Practice

**Good news: You don't need to implement BPE yourself!**

**Popular libraries:**

**1. Hugging Face Tokenizers (Fast Rust implementation):**

```
from tokenizers import Tokenizer
from tokenizers.models import BPE
from tokenizers.trainers import BpeTrainer

tokenizer = Tokenizer(BPE())
trainer = BpeTrainer(vocab_size=30000, special_tokens=["<PAD>", "<UNK>"])
tokenizer.train(files=["corpus.txt"], trainer=trainer)

# Use it
output = tokenizer.encode("unhappiness")
print(output.tokens)  # ['un', 'happi', 'ness']
```

# Implementing BPE in Practice (continued)

## 2. SentencePiece (Google):

```
import sentencepiece as spm

# Train SentencePiece model
spm.SentencePieceTrainer.train(
    input='corpus.txt',
    model_prefix='tokenizer',
    vocab_size=30000,
    model_type='bpe'  # or 'unigram'
)

# Load and use
sp = spm.SentencePieceProcessor()
sp.load('tokenizer.model')
tokens = sp.encode_as_pieces("unhappiness")
print(tokens)  # ['un', 'happi', 'ness']
```

## 3. tiktoken (OpenAI - for GPT models):

```
import tiktoken

enc = tiktoken.get_encoding("cl100k_base")  # GPT-4 encoding
tokens = enc.encode("unhappiness")
print([enc.decode_single_token_bytes(t) for t in tokens])
```

# Common Pitfalls and Best Practices

## Common Misconception

**"I should train a new tokenizer for every task"**
Usually FALSE! Use pre-trained tokenizers when fine-tuning models. Only train new tokenizer if: (1) New language not in pre-trained vocab, or (2) Highly specialized domain (medical, legal) with unique terminology

**Best Practices:**

1. **Vocabulary size:** Start with 30K-50K (industry standard)
2. **Special tokens:** Always include PAD, UNK, BOS, EOS tokens
3. **Normalization:** Decide on casing (lowercase vs mixed case)
4. **Pre-tokenization:** For English/European languages, split on spaces first
5. **Corpus representativeness:** Train on data similar to inference data

## Real-World Application

**Real mistake:** A company trained a tokenizer on formal business emails, then deployed it for social media text. Result: Poor performance due to emoji, slang, abbreviations being split into too many tokens. Lesson: Match training data to use case!

# Summary: The Tokenization Journey

**What we learned today:**

1. **The Problem:** Word-level tokenization fails due to OOV (out-of-vocabulary)
2. **The Solution:** Subword tokenization (BPE, WordPiece, SentencePiece)
3. **The Algorithm:** BPE iteratively merges most frequent adjacent pairs
4. **The Advantage:** Rare words decompose into meaningful subword units
5. **The Trade-off:** Vocabulary size balances efficiency vs granularity
6. **The Impact:** Tokenization significantly affects model performance

**Key Insight:** Tokenization is not a preprocessing afterthought - it's a fundamental design choice that affects model architecture, training, and performance!

# Final Checkpoint: Complete Understanding

**Checkpoint**

**Final self-assessment:**

Can you explain these to a friend?

- Why does GPT-4 split "unhappiness" into [un][happi][ness]?
- What's the difference between BPE and WordPiece?
- Why do multilingual models need larger vocabularies?
- How does tokenization affect training speed?

If you can answer these, you understand tokenization!

**Challenge:** Try encoding "supercalifragilisticexpialidocious" with BPE. How many subword units do you think it would produce? (Answer: Depends on vocabulary, but typically 8-12 units that capture phonetic patterns)

# Looking Ahead: Week 9

**Now that we have tokens, how do we generate text?**

**Next week: Decoding Strategies**

- Greedy decoding (simple but flawed)
- Beam search (better but still limited)
- Sampling methods (temperature, top-k, top-p)
- Why do chatbots sometimes repeat themselves?
- How to control creativity vs coherence

**Real-World Application**

**Teaser:** Have you noticed ChatGPT sometimes gives boring responses and sometimes creative ones? That's decoding strategy at work! Next week we'll learn how to control this.

**See you next week!**