Tokenization Methods

Tokenization is the process of breaking text into smaller units (tokens) that can be processed by NLP models. Different methods balance vocabulary size, sequence length, and language coverage.

🧱 1. Word-level Tokenization

Each unique word in the corpus is a token.

```
Input: unhappiness
Tokens: [ "unhappiness" ]
```

✅ Simple

❌ OOV problem – if “unhappiness” never appeared in training, it cannot be represented (Out-of-Vocabulary).

🔡 2. Character-level Tokenization

Each character is treated as a token.

```
Input: unhappiness
Tokens: [ "u", "n", "h", "a", "p", "p", "i", "n", "e", "s", "s" ]
```

Vocab: 26 letters (plus punctuation, digits, etc.)

✅ No OOV problem

❌ Very long sequences → harder for models to capture meaning efficiently.

🧩 3. Subword-level Tokenization

BPE (Byte Pair Encoding)

```
•   Iteratively merge the most frequent pair of tokens (starting from characters).
•   Creates new tokens for frequent patterns (“th”, “ing”, “tion”…).
•   Deterministic: same input → same tokenization.
•   Used in GPT-2, GPT-3, RoBERTa, etc.
•   ⚙ Frequency-based, not probabilistic.
```

Example corpus:

```
unhappiness happiness unhappy
```

Step 1: Break into characters

```
u n h a p p i n e s s
h a p p i n e s s
u n h a p p y
```

Step 2: Count pairs (frequencies)

Pair	Count
h a	3
a p	3
p p	3
...	...

Top pair = p p → merge into "pp".

Step 3: Repeat merges

Now we have tokens like happ, ppiness, etc.

Next merges: ("h app" → "happ"), ("iness" → "iness"), etc.

After several merges, the vocabulary might contain:

```
["un", "happ", "iness", "happy"]
```

Step 4: Tokenize new word "unhappiness"

```
Input: unhappiness
Tokens: [ "un", "happ", "iness" ]
```

✅ Frequent subwords merged → efficient

✅ No OOV (compositional)

⚙️ Deterministic and frequency-based.

WordPiece

```
•   Similar to BPE but uses likelihood improvement instead of raw
frequency.
•   At each step, choose merge that maximizes training corpus likelihood.
•   Used in BERT, DistilBERT, etc.
•   Slightly more sophisticated than BPE.
```

At each iteration, compute improvement in corpus likelihood $p(\text{corpus}) = \prod_i p(\text{token}_i)$.

```
Choose the merge that gives the biggest improvement in log-likelihood
(i.e., best compression of corpus probability).
```

```
"un" 500, "happy" 1000, "iness" 400
"unhappiness" 0
```

```
•   Merge “un” + “happy” → “unhappy” gives a big probability boost.
•   Merge “unhappy” + “ness” → “unhappiness” appears rarely, gives smaller
boost.
```

So WordPiece keeps:

```
["un", "happy", "ness"]
```

score=(freq_of_pair)/(freq_of_first_element×freq_of_second_element)

✅ Merges guided by probabilistic likelihood

✅ Keeps subwords that most improve modeling efficiency

Byte-level

```
•   Variant of BPE that operates on raw bytes (0–255) instead of Unicode
characters.
•   Used in GPT-2, GPT-3, GPT-4.
•   Base alphabet = 256 bytes, not 26 characters.
```

Why bytes?

```
•   Works for any language and encoding (UTF-8 safe).
•   No unknown tokens (<unk> never needed).
•   Reversible (decode bytes → exact original text).
```

```
"unhappiness" → [117, 110, 104, ...]
```

Frequent byte pairs are merged, just like BPE.

✅ Universal (handles emojis, non-English text) ✅ Fully reversible ✅ Compact and encoding-agnostic ⚙️ Base alphabet = 256 bytes, not 26 characters.

```
•   No unknown tokens (<unk> never needed)
•   Simple reversible encoding (decode bytes → exact original text)
```

Aspect	Character-level BPE	Byte-level BPE
Base alphabet	26–200 characters	256 bytes (0–255)
Encoding required	Yes (Unicode-sensitive)	No (encoding-agnostic)
Language coverage	English or Latin scripts	All languages
Vocabulary size	Larger	More compact
Robustness	May fail on unseen chars	Never fails (no)
Used in	Early subword tokenizers	GPT-2, GPT-3, GPT-4

📊 5. Entropy-based / Unigram LM Tokenization

Unigram Language Model

```
•   Starts with a large set of candidate subwords.
•   Learns a probabilistic model over them.
•   Iteratively removes tokens that least affect total likelihood (entropy
minimization).
•   Used in SentencePiece (e.g., ALBERT, T5).
•   Non-deterministic: can sample multiple tokenizations.
```

```
Tokenizations possible:
1. ["un", "happiness"] → p("un") * p("happiness")
2. ["un", "happy", "ness"] → p("un") * p("happy") * p("ness")
```

If (2) has higher probability → that's the chosen segmentation.

✅ Flexible, probabilistic

✅ Allows multiple tokenizations with sampling (for data augmentation)

Sentence Piece Framework

```
•   A framework (not a single algorithm) supporting BPE or Unigram LM.
•   Operates directly on raw text without whitespace.
•   Works on any language, including agglutinative or multilingual
corpora.
•   Used in mT5, XLM-R.
```

Entropy-based tokenization

```
Some research methods explicitly optimize information entropy:
•   Minimize average code length or maximize compression rate (related to
Shannon entropy).
•   Unigram LM is a probabilistic example of this idea.
•   Useful for designing adaptive tokenizers for multilingual LLMs.
```

Find token boundaries that minimize total encoding entropy:

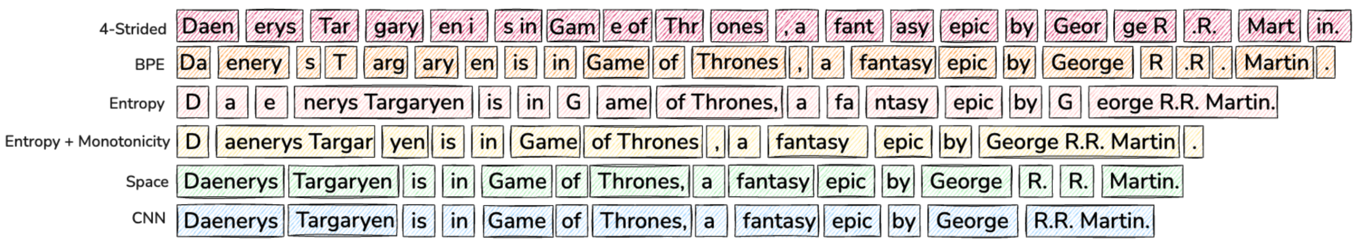
$$H = -\sum_{t} p(t) \log p(t)$$

You choose tokens that yield smallest average code length over corpus (Shannon-optimal).

The Unigram LM implicitly does this — pruning tokens that don't improve encoding efficiency.

✅ Theoretically optimal (compression perspective)

✅ Adaptive to corpus and multilingual text



Questions

Q1. Explain the main trade-offs between word-level, character-level, and subword-level tokenization.

When would you prefer each, and why?

```
•   Word-level tokenization treats each word as an atomic token.
•   ✅ Pros: Simple and interpretable.
•   ❌ Cons: Has a severe OOV problem — unseen words in inference cannot
be represented.
•   Best for: Closed-vocabulary tasks, e.g., domain-specific corpora
(legal or medical).
•   Character-level tokenization uses each character as a token.
•   ✅ Pros: Completely eliminates OOV; smallest possible vocabulary.
•   ❌ Cons: Very long sequences → harder for models to learn long-range
dependencies.
•   Best for: Languages with small alphabets (Korean, English), or low-
resource scenarios.
•   Subword-level tokenization (e.g., BPE, WordPiece, Unigram LM) balances
both.
•   ✅ Pros: Reduces OOV by decomposing rare words into known subwords.
•   ❌ Cons: Some ambiguity at token boundaries.
•   Best for: Modern LLMs — captures both morphological and semantic
structure efficiently.
```

Q2. Describe the Byte Pair Encoding (BPE) algorithm.

How are merge operations determined?

Why does BPE help reduce the Out-of-Vocabulary (OOV) problem?

```
•   BPE starts from individual characters and iteratively merges the most
frequent adjacent pairs in the corpus.
•   Merges are purely frequency-based — each step finds the most common
pair and replaces it with a new token.
•   Over time, this forms frequent subwords (e.g., “un” + “happy” →
“unhappy”).
```

Why it helps OOV:

Any unseen word (e.g., “unfriendliness”) can still be represented as a combination of known subwords (“un”, “friend”, “liness”). Thus, no unknown token is required — the model can always decompose text into valid tokens.

Q3. How does WordPiece differ from BPE?

What does it optimize during training, and what’s the advantage of this difference?

Both start with character-level vocabularies and merge subwords.

Difference: • BPE merges based on raw frequency. • WordPiece merges based on maximum likelihood improvement:

$$\text{Choose merge that maximimizes} p(\text{corpus}) = \prod_i p(\text{token}_i)$$

So each merge is chosen to increase corpus log-likelihood the most.

Advantage:

```
•   This probabilistic criterion leads to a vocabulary that better
reflects language statistics, not just string frequency.
•   It avoids overly merging common substrings that don't improve modeling
efficiency.
•   Used in BERT and DistilBERT for more balanced subword coverage.
```

Q4. What is the motivation behind using byte-level BPE (as in GPT-2/3/4)?

Why does it use a 256-byte alphabet instead of characters?

What problem does this solve for multilingual or emoji text?

This approach treats all text as raw UTF-8 bytes, making it:

```
•   ✅ Encoding-agnostic — no need for Unicode normalization.
•   ✅ Language-independent — any language, symbol, or emoji can be
represented as bytes.
•   ✅ Reversible — decoding the bytes always reconstructs the original
text.
```

Why 256 bytes, not 26 characters:

Every symbol (including emoji and Chinese characters) can be encoded using bytes. This ensures no OOV tokens — <unk> is unnecessary.

Thus, byte-level BPE provides universality and robustness, essential for multilingual LLMs like GPT-2, 3, and 4.

Q5. How does the Unigram Language Model (Unigram LM) tokenizer work?

What objective function does it optimize, and how is entropy involved?

Why is it considered probabilistic and non-deterministic?

```
•   Starts with a large candidate vocabulary (all substrings).
•   Trains a probabilistic model assigning each token a probability p(t).
•   The likelihood of a corpus is:
```

$$\mathrm{P(corpus)} = \prod_{\mathrm{sentence}} \sum_{\mathrm{tokenizations}} \prod_{t \in \mathrm{tokens}} p(t)$$

• Tokens that contribute least to the overall likelihood are removed iteratively, minimizing total encoding entropy:

$$H = -\sum_t p(t) \log p(t)$$

• The model chooses the tokenization that maximizes

$$\prod_t p(t)$$

Probabilistic nature:

Different tokenizations may have similar likelihoods → the model can sample among them.

✅ Used in SentencePiece (T5, ALBERT)

✅ Provides flexibility, multilingual coverage, and entropy-based optimization.

Q6. Compare the probabilistic and frequency-based tokenization methods.

How does the notion of likelihood or information entropy affect token merges?

Aspect	Frequency-based (BPE)	Probabilistic (WordPiece / Unigram LM)
Merge Criterion	Most frequent adjacent pairs	Maximizes corpus likelihood
Determinism	Deterministic	Probabilistic (can sample)
Theoretical Basis	Count statistics	Information theory (entropy)
Adaptivity	May overfit frequent patterns	Balances global token efficiency
Example Models	GPT-2, RoBERTa	BERT, T5, ALBERT

Q7. Explain the relationship between entropy minimization and tokenization efficiency.

Why does minimizing entropy lead to better compression or model performance?

```
•   Entropy measures the average information per token:
```

```
```

$H = -\sum_t p(t) \log p(t)$

```
•   Lower entropy means the model represents the corpus with fewer bits on
average (better compression).
•   A tokenizer that minimizes entropy creates tokens that efficiently
encode frequent patterns, maximizing reuse while minimizing redundancy.
```

Hence, entropy minimization aligns with:

```
•   Compact vocabularies
•   Shorter encoded sequences
•   Efficient downstream learning
```

This is the theoretical motivation behind Unigram LM and entropy-based tokenizers.

Q8. In what way is SentencePiece a framework rather than a single algorithm?

How can it unify BPE and Unigram LM tokenization approaches?

```
•   SentencePiece is a general framework for tokenization that can
implement either BPE or Unigram LM (or others).
•   Works directly on raw text without whitespace segmentation, making it
suitable for languages without spaces (e.g., Japanese, Chinese).
•   Provides tools for:
•   Unicode normalization
•   Training tokenizers from scratch
•   Sampling multiple tokenizations for augmentation
```

✅ Used in multilingual models like mT5, XLM-R.

✅ Supports unsupervised, language-agnostic text segmentation.

Q9. Suppose you train a multilingual model on English, Chinese, and Arabic.

Which tokenization approach would you choose and why? What challenges would arise if you used word-level or character-level tokenization?

```
•   Byte-level BPE or Unigram LM via SentencePiece would be ideal.
•   Reasoning:
```

```
•   Word-level tokenization fails (different scripts, no consistent
whitespace).
•   Character-level tokenization would produce very long sequences for
Chinese and Arabic.
•   Byte-level BPE covers all languages uniformly using 256-byte units.
•   Unigram LM learns efficient subwords probabilistically and handles
multilingual distributions well.
```

Best practice:

Use SentencePiece-Unigram LM with a shared multilingual vocabulary — efficient, compact, and robust across scripts.

Q10. Why does a byte-level tokenizer ensure that (unknown token) is never needed?

How does this property benefit large-scale generative models like GPT?

```
•   In byte-level BPE, every character or symbol is encoded as a sequence
of bytes (0–255).
•   Since every possible byte exists in the vocabulary, any string (even
unseen or corrupted) can be represented.
```

→ No token is required.

Benefits for GPT models:

```
•   Perfect reversibility (no information loss).
•   Seamless support for any language, emoji, or binary input.
•   Robust to unseen words or typos — essential for open-domain text
generation.
```

Hence, GPT’s tokenizer is universal, compact, and lossless, ensuring the model can always produce valid decodable outputs.