# Lecture 2.2 - Tokens and Tokenization

Generative AI Teaching Kit

# This lecture

- Introduction to Tokens
- Tokenization approaches
- LLM Limitations from tokenization
- Token-free models

# Introduction to Tokens

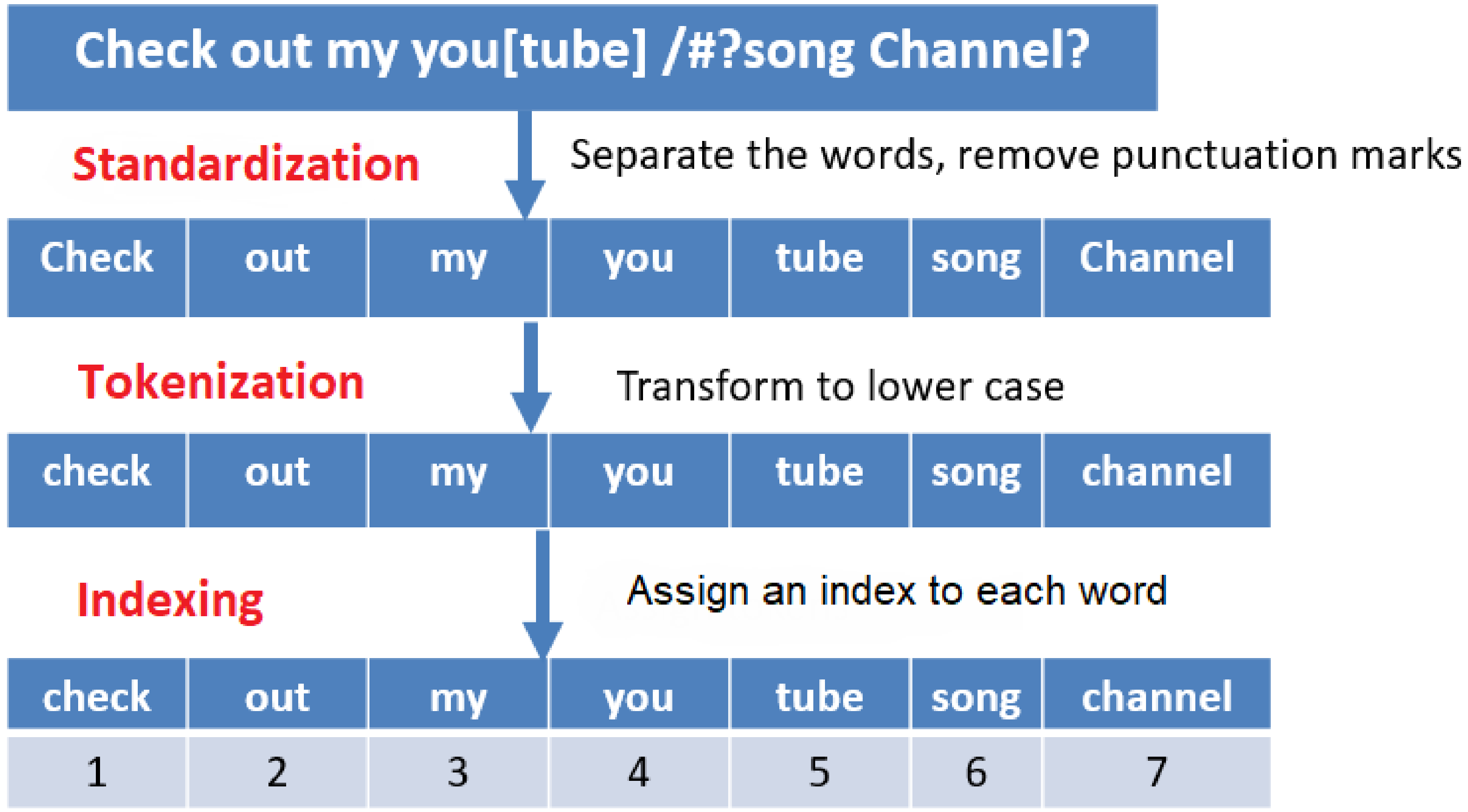
# What are tokens?

Tokens are the building blocks of all natural language tasks and language models.

They represent language elements, like words or characters, or parts of words, which can be stored as a key-value pairing, to enable the expression of text as a sequence of numbers.

Language can be processed or encoded as a series of tokens IDs, and then these IDs, once selected from a language model, can be decoded back to language elements for human interpretation.

How these tokens are built for a given language, or dataset varies and can have a drastic impact on the performance and utility of the language models



Check out my you[tube] /#?song Channel?

**Standardization** Separate the words, remove punctuation marks

| Check | out | my | you | tube | song | Channel |
|---|---|---|---|---|---|---|

**Tokenization** Transform to lower case

| check | out | my | you | tube | song | channel |
|---|---|---|---|---|---|---|

**Indexing** Assign an index to each word

| check | out | my | you | tube | song | channel |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

# Important terms

When discussing tokens and tokenization, it is important to consider some terms that describe how a natural language phase is tokenized.

If we take the following phrase, or **sequence,** we need to use our **vocabulary**, to split it into separate **tokens.**

The moon, Earth's only natural satellite, has been a subject of fascination and wonder for thousands of years.

**Token**: Basic building block

- The
- Moon
- ,
- Earth’s
- Only
- …..
- years

**Sequence**: Sequential list of tokens

- The moon,
- Earth’s only natural satellite
- Has been a subject of
- ….
- Thousands of years

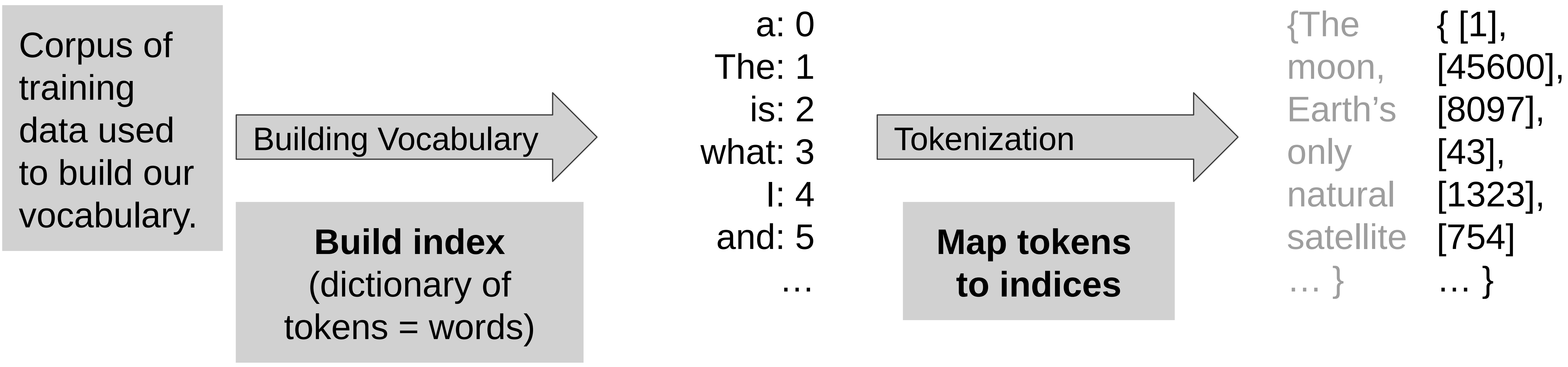
**Vocabulary**: Complete list of known tokens

```
{
    1:"The",
    569:"moon",
    122: ",",
    430:"Earth",
    50:"**’s",
…}
```

# Tokenization Methods

# Tokenization: Word-level

The moon, Earth's only natural satellite, has been a subject of fascination and wonder for thousands of years.



Corpus of training data used to build our vocabulary.

Building Vocabulary

**Build index** (dictionary of tokens = words)

a: 0
The: 1
is: 2
what: 3
I: 4
and: 5
…

Tokenization

**Map tokens to indices**

| | |
|---|---|
| {The | { [1], |
| moon, | [45600], |
| Earth’s | [8097], |
| only | [43], |
| natural | [1323], |
| satellite | [754] |
| … } | … } |

Pros

Intuitive.

Quick and simple to implement

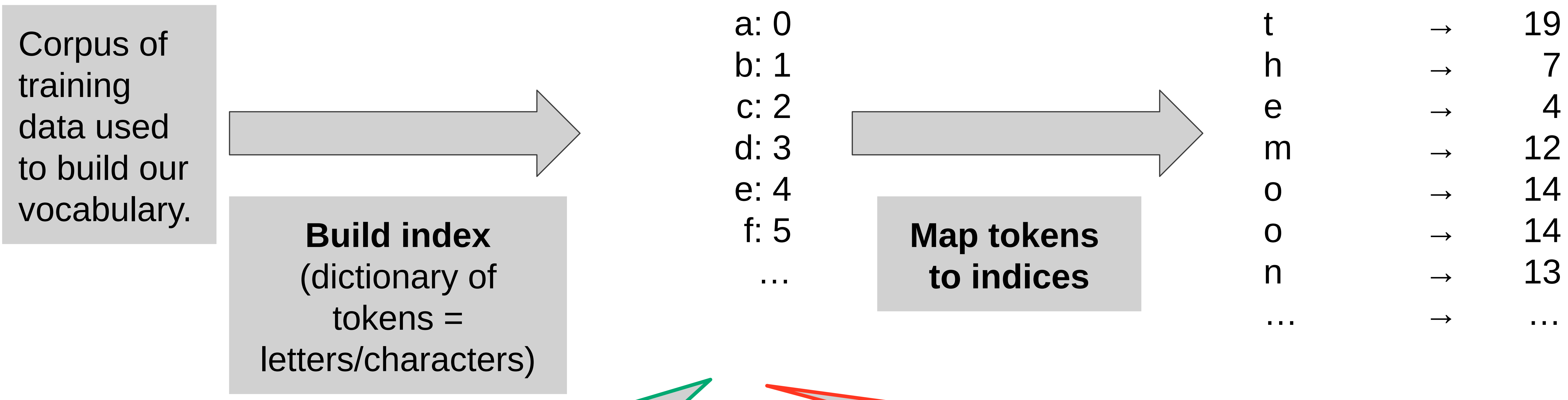
Cons

**Big** vocabularies.

Complications such as handling misspellings and other out-of-vocabulary words.

# Tokenization: Character-level

The moon, Earth's only natural satellite, has been a subject of fascination and wonder for thousands of years.



Corpus of training data used to build our vocabulary.

**Build index** (dictionary of tokens = letters/characters)

a: 0
b: 1
c: 2
d: 3
e: 4
f: 5
…

**Map tokens to indices**

| | | |
|---|---|---|
| t | → | 19 |
| h | → | 7 |
| e | → | 4 |
| m | → | 12 |
| o | → | 14 |
| o | → | 14 |
| n | → | 13 |
| … | → | … |

Pros

Small vocabulary.

No out-of-vocabulary words.

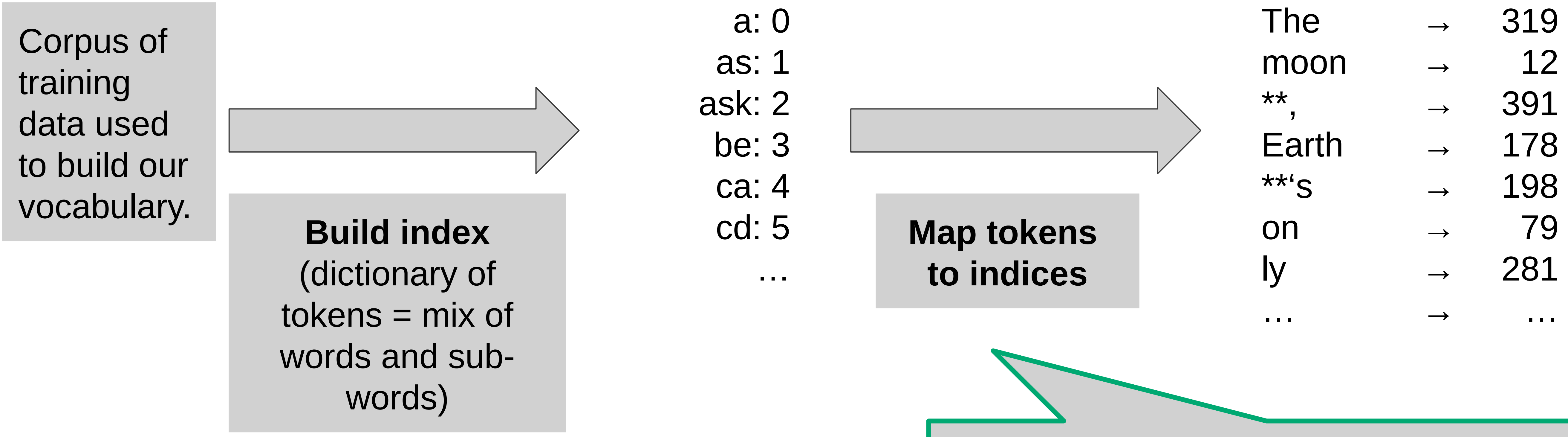
Cons

Loss of context within words.

Much longer sequences for a given input.

# Tokenization: Sub-word-level

The moon, Earth's only natural satellite, has been a subject of fascination and wonder for thousands of years.



Corpus of training data used to build our vocabulary.

**Build index** (dictionary of tokens = mix of words and sub-words)

a: 0
as: 1
ask: 2
be: 3
ca: 4
cd: 5
…

**Map tokens to indices**

| | | |
|---|---|---|
| The | → | 319 |
| moon | → | 12 |
| **, | → | 391 |
| Earth | → | 178 |
| **‘s | → | 198 |
| on | → | 79 |
| ly | → | 281 |
| … | → | … |

**Byte Pair Encoding (BPE) a popular encoding.**

Start with a small vocab of characters.

Iteratively merge frequent pairs into new bytes in the vocab (such as “b”,”e” → “be”).

<u>Compromise</u>

“Smart” vocabulary built from characters which co-occur frequently.

More robust to novel words.

# Tokenization

| Tokenization method | Tokens | Token count | Vocab size |
|---|---|---|---|
| Sentence | ‘The moon, Earth's only natural satellite, has been a subject of fascination and wonder for thousands of years.’ | 1 | # sentences in doc |
| Word | 'The', 'moon,', "Earth's", 'only', 'natural', 'satellite,', 'has', 'been', 'a', 'subject', 'of', 'fascination', 'and', 'wonder', 'for', 'thousands', 'of', 'years.' | 18 | 171K (English[1]) |
| **Sub-word** | 'The', 'moon', ',', 'Earth', "'", 's', 'on', 'ly', 'n', 'atur', 'al', 's', 'ate', 'll', 'it', 'e', ',', 'has', 'been', 'a', 'subject', 'of', 'fascinat', 'ion', 'and', 'w', 'on', 'd', 'er', 'for', 'th', 'ous', 'and', 's', 'of', 'y', 'ears', '.' | 37 | (varies) |
| Character | 'T', 'h', 'e', ' ', 'm', 'o', 'o', 'n', ',', ' ', 'E', 'a', 'r', 't', 'h', "'", 's', ' ', 'o', 'n', 'l', 'y', ' ', 'n', 'a', 't', 'u', 'r', 'a', 'l', ' ', 's', 'a', 't', 'e', 'l', 'l', 'i', 't', 'e', ',', ' ', 'h', 'a', 's', ' ', 'b', 'e', 'e', 'n', ' ', 'a', ' ', 's', 'u', 'b', 'j', 'e', 'c', 't', ' ', 'o', 'f', ' ', 'f', 'a', 's', 'c', 'i', 'n', 'a', 't', 'i', 'o', 'n', ' ', 'a', 'n', 'd', ' ', 'w', 'o', 'n', 'd', 'e', 'r', ' ', 'f', 'o', 'r', ' ', 't', 'h', 'o', 'u', 's', 'a', 'n', 'd', 's', ' ', 'o', 'f', ' ', 'y', 'e', 'a', 'r', 's', '.' | 110 | 52 + punctuation (English) |

# Byte-Pair Encoding

**Byte Pair Encoding (BPE)** is a sub-word tokenization algorithm used to break text into smaller units (sub-words) to handle rare words and unknown tokens effectively. It’s widely used in NLP models like GPT and BERT.

**Steps in the BPE Algorithm**

1. **Initialization:**
Start with a corpus where each word is represented as a sequence of characters (with an end-of-word marker, e.g., </w>).

2. **Count Pair Frequencies:**
Count the frequency of every adjacent character pair in the corpus.

3. **Merge the Most Frequent Pair:**
Find the most frequent pair and merge it into a new token.

4. **Update Corpus and Repeat:**
Update the corpus with the newly created sub-word tokens and repeat steps 2–3 until you reach the desired number of merges.

5. **Build the Final Vocabulary:**
After merges, the resulting vocabulary contains the original characters plus newly formed sub-word tokens.

Example Step 1:
"low" -> ["l", "o", "w", "</w>"]
"lower" -> ["l", "o", "w", "e", "r", "</w>"]

Example Step 2:
Corpus: ["l", "o", "w", "</w>"], ["l", "o", "w", "e", "r", "</w>"]
Pair frequencies:
("l", "o") → 2
("o", "w") → 2
("w", "</w>") → 2
("o", "e") → 1
("e", "r") → 1

Example Step 3:
Merge ("l", "o") → ["lo", "w", "</w>"], ["lo", "w", "e", "r", "</w>"].

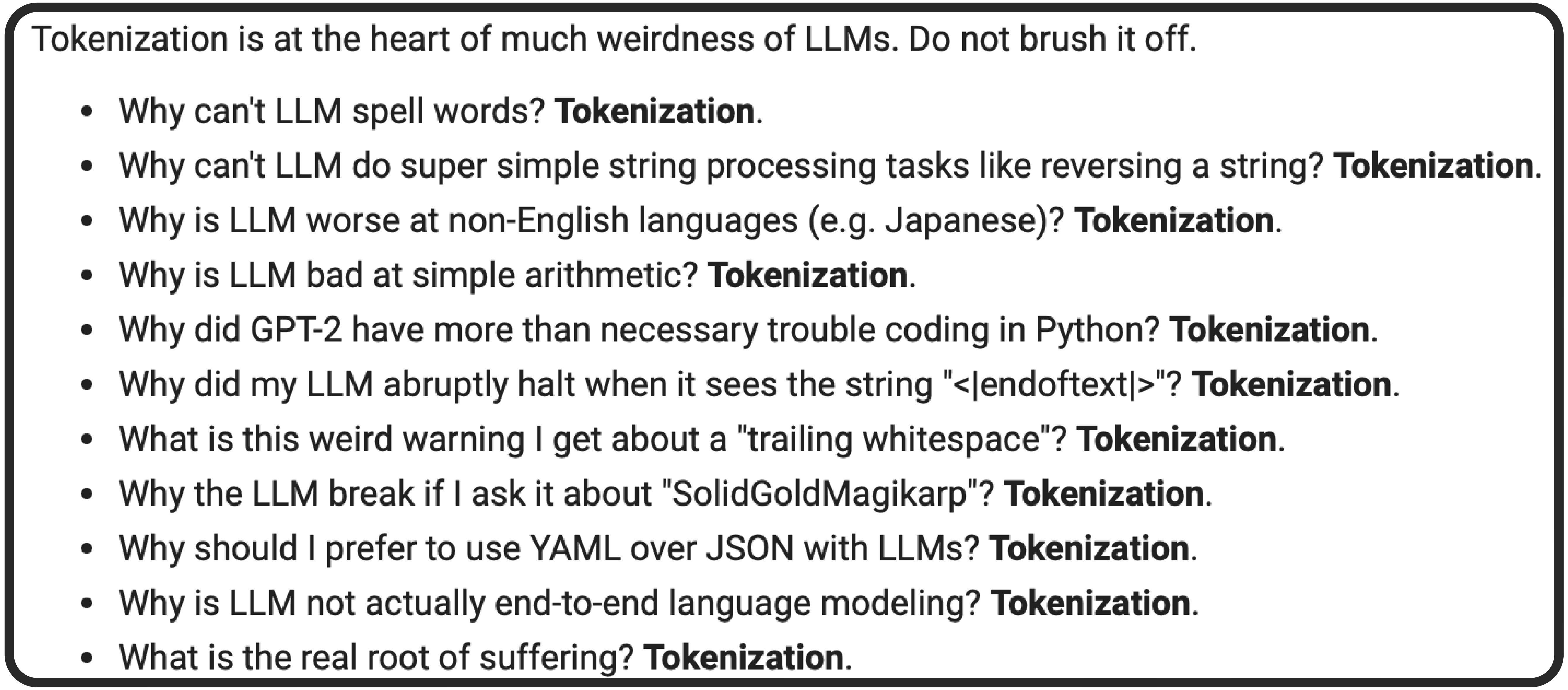
**Advantages of BPE**

- **Handles Rare Words:** Breaks rare or unknown words into sub-word units, avoiding out-of-vocabulary (OOV) issues.
- **Efficient Vocabulary Size:** Produces a compact vocabulary by balancing characters and frequent sub-words.
- **Language Agnostic:** Works well across different languages.

# Tokenization and Model Behavior

# Effect of Tokenization

Large Language Models often have several issues that affect the performance and applicability to various talks. Many of these, highlighted below in a talk from Andrej Karpathy, are a direct results of tokenization:



Tokenization is at the heart of much weirdness of LLMs. Do not brush it off.

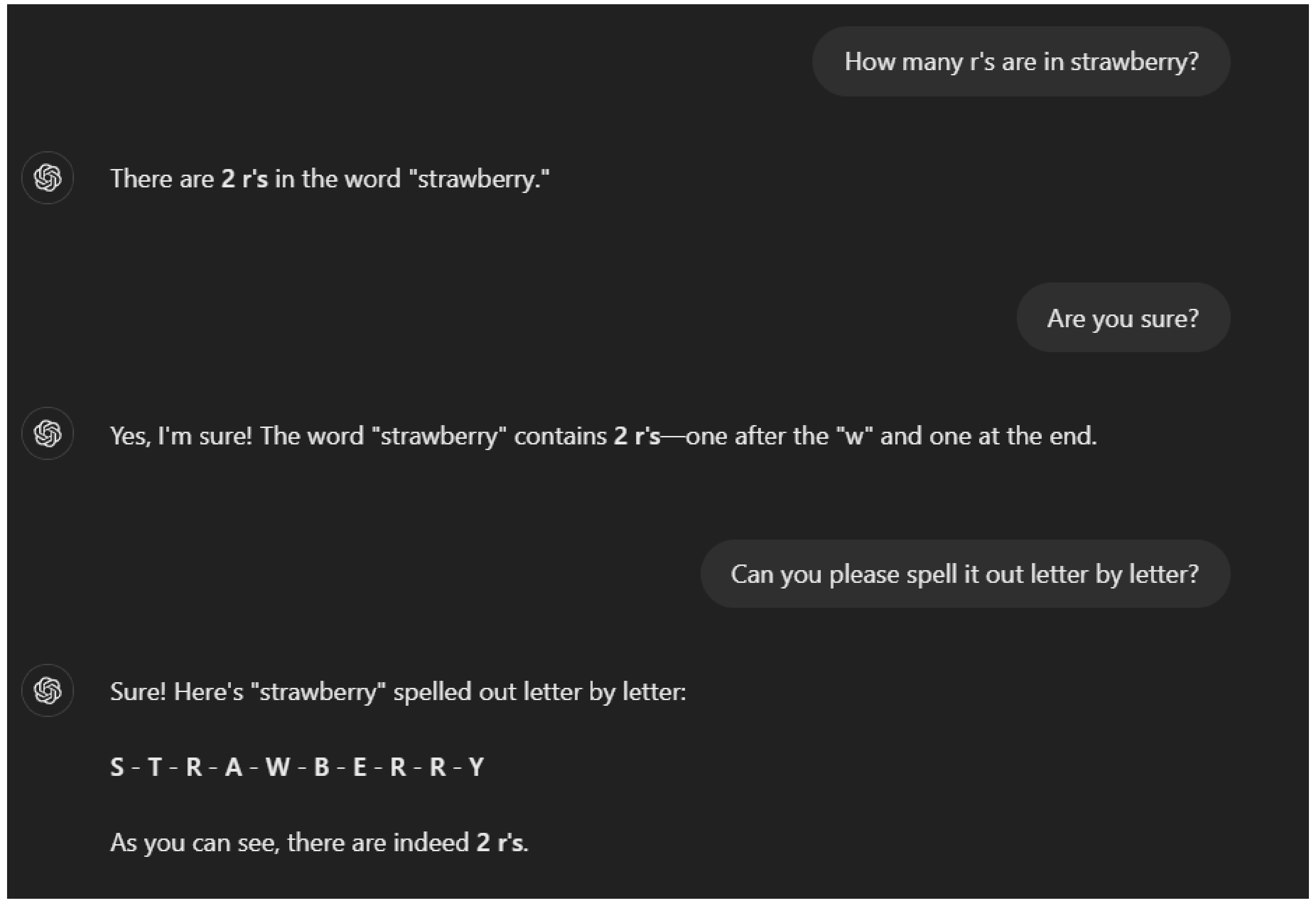
- Why can't LLM spell words? **Tokenization**.
- Why can't LLM do super simple string processing tasks like reversing a string? **Tokenization**.
- Why is LLM worse at non-English languages (e.g. Japanese)? **Tokenization**.
- Why is LLM bad at simple arithmetic? **Tokenization**.
- Why did GPT-2 have more than necessary trouble coding in Python? **Tokenization**.
- Why did my LLM abruptly halt when it sees the string "<|endoftext|>"? **Tokenization**.
- What is this weird warning I get about a "trailing whitespace"? **Tokenization**.
- Why the LLM break if I ask it about "SolidGoldMagikarp"? **Tokenization**.
- Why should I prefer to use YAML over JSON with LLMs? **Tokenization**.
- Why is LLM not actually end-to-end language modeling? **Tokenization**.
- What is the real root of suffering? **Tokenization**.

# Effect of Tokenization – Spelling and Char-level issues

**1. LLMs Struggle with Spelling and Rare Words:**
Tokenization splits words into sub-words (e.g., “unpredictable” → [“un”, “predict”, “able”]).
Misspelled words (e.g., “teh”) fragment into low-probability sub-words, causing errors.

**2. String Manipulation Is Difficult:**
LLMs operate at the token level, not character level.
Tasks like reversing strings (“hello” → “olleh”) require access to individual characters, which tokenization obscures.

**3. Granularity Mismatch:**
Sub-word tokenization prioritizes meaning over form, leading to poor performance on character-sensitive tasks.
Why It Happens:
LLMs are trained to understand semantics, not fine-grained character-level details.



How many r's are in strawberry?

There are **2 r's** in the word "strawberry."

Are you sure?

Yes, I'm sure! The word "strawberry" contains **2 r's**—one after the "w" and one at the end.

Can you please spell it out letter by letter?

Sure! Here's "strawberry" spelled out letter by letter:

**S - T - R - A - W - B - E - R - R - Y**

As you can see, there are indeed **2 r's**.

The “strawberry problem”

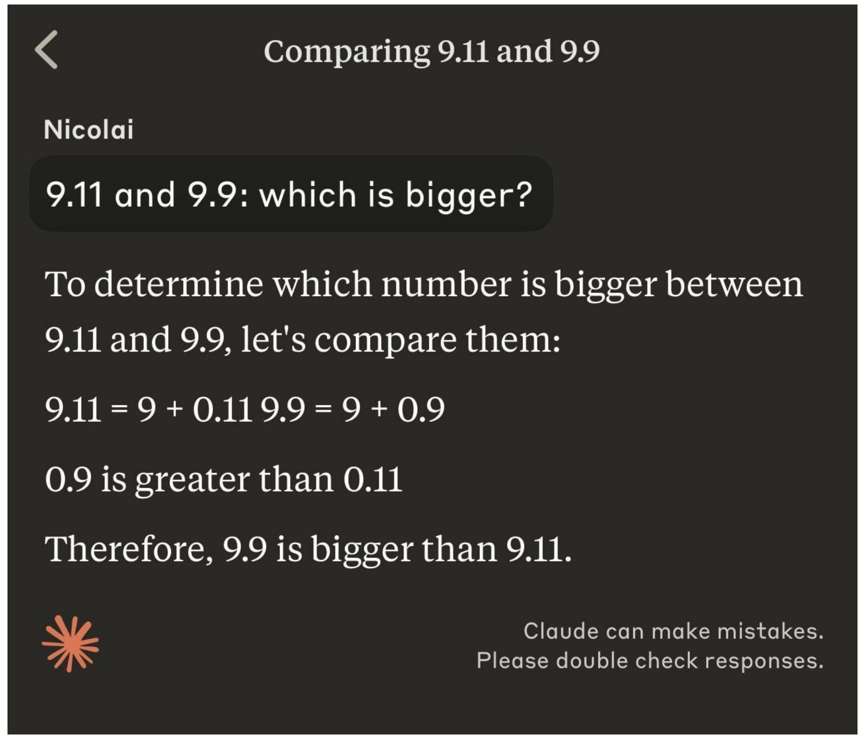
# Effect of Tokenization – Arithmetic Challenges

**1. LLMs Learn Patterns, Not Rules:**
LLMs are trained on text data to predict the next token, not to understand or apply mathematical rules.
Arithmetic operations require explicit logic, not just pattern recognition.

**2. Tokenization Can Cause Numerical Errors:**
Numbers may be split into multiple tokens, especially for large or complex numbers (e.g., “123456” → [“123”, “456”]).
Consequence:
Operations requiring precise numerical understanding are prone to failure.

**3. Training Data Limitations:**
LLMs are exposed to math through text (e.g., “5 + 3 = 8”) rather than structured numeric computation.
Models learn approximations based on seen examples, not systematic calculations.

**4. No Internal Math Engine:**
LLMs do not inherently perform arithmetic. Their “calculations” are derived from patterns observed in training data, leading to inconsistent results.



Comparing 9.11 and 9.9

Nicolai

9.11 and 9.9: which is bigger?

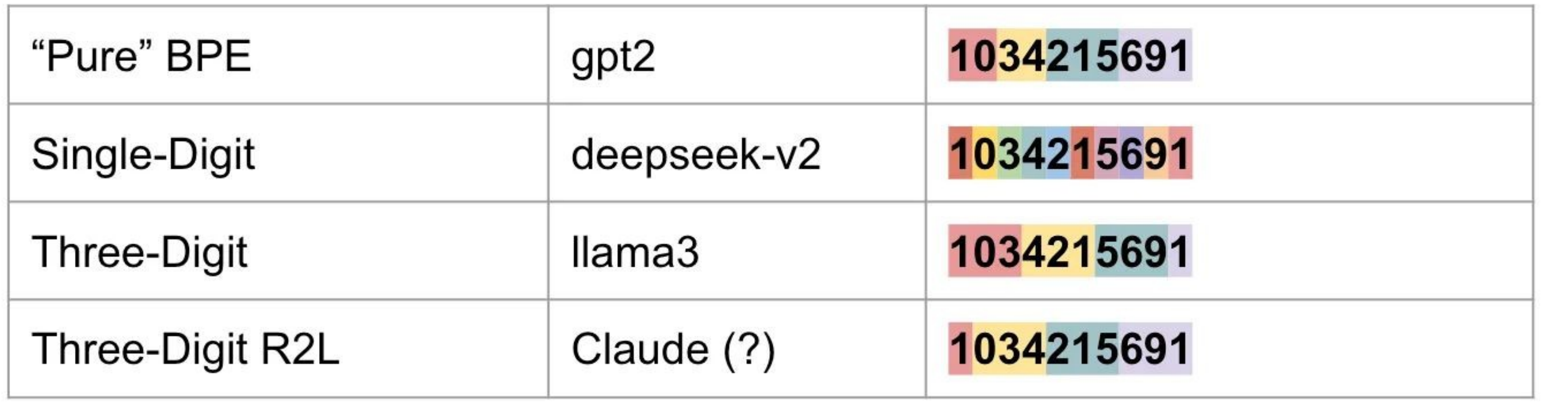
To determine which number is bigger between 9.11 and 9.9, let's compare them:

9.11 = 9 + 0.11 9.9 = 9 + 0.9

0.9 is greater than 0.11

Therefore, 9.9 is bigger than 9.11.

Claude can make mistakes.
Please double check responses.



| | | |
|---|---|---|
| “Pure” BPE | gpt2 | 1034215691 |
| Single-Digit | deepseek-v2 | 1034215691 |
| Three-Digit | llama3 | 1034215691 |
| Three-Digit R2L | Claude (?) | 1034215691 |

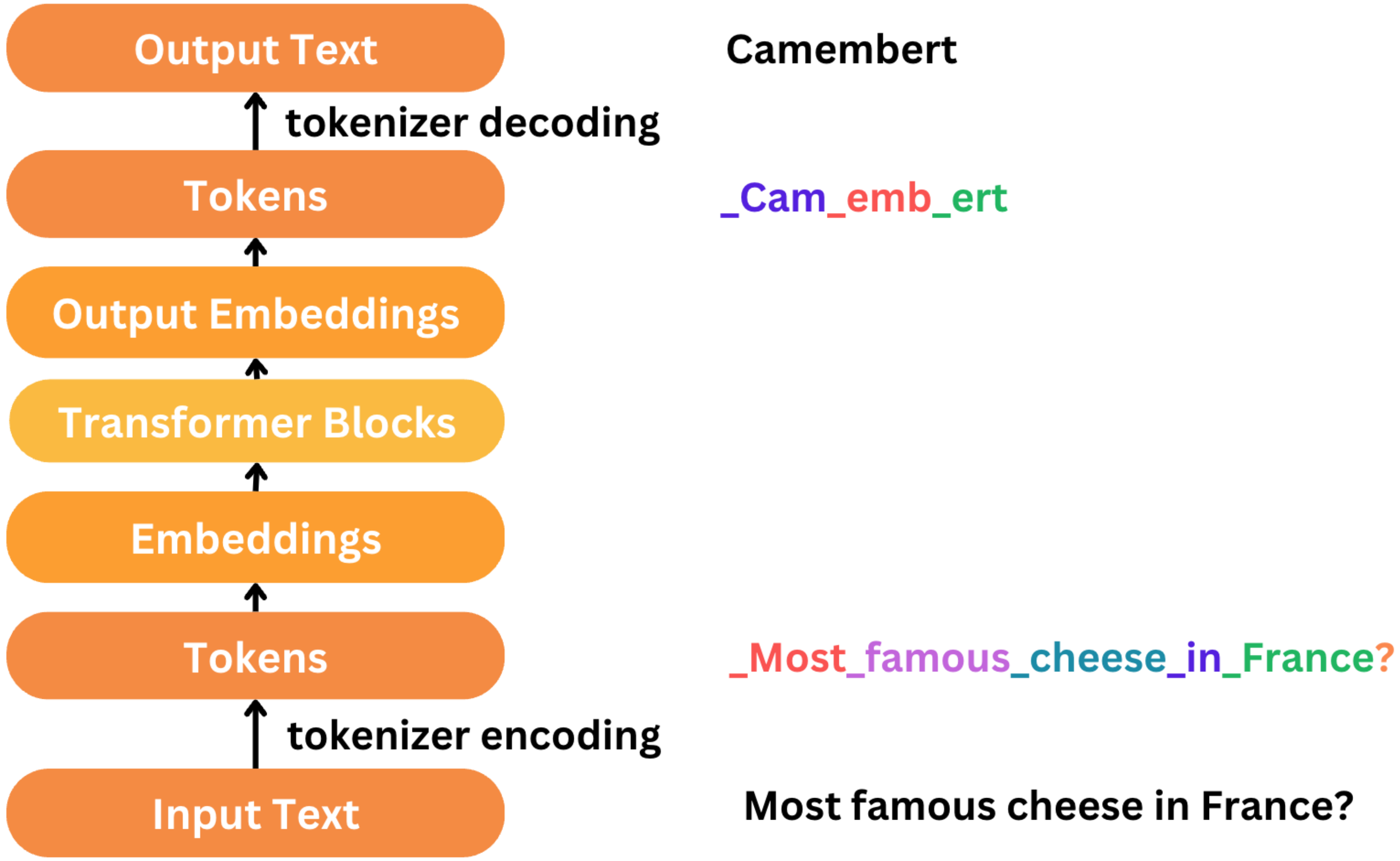
# Effect of Tokenization – LLMs as systems

**1. Dependence on Tokenization:**
LLMs require preprocessing (e.g., tokenization) to handle input text.

**2. Lack of Multimodal Capabilities:**
Cannot process raw audio, images, or other non-text data directly without external tools.

**3. Reliance on External Components:**
Needs integration with tools for tasks like speech recognition, image understanding, or advanced reasoning.

**4. No Built-In Execution:**
Cannot perform exact computations or symbolic reasoning without external plugins.



# Token-free and modern approaches

# Challenges with Tokenization-based Models

**Fixed Vocabulary Limitations:**

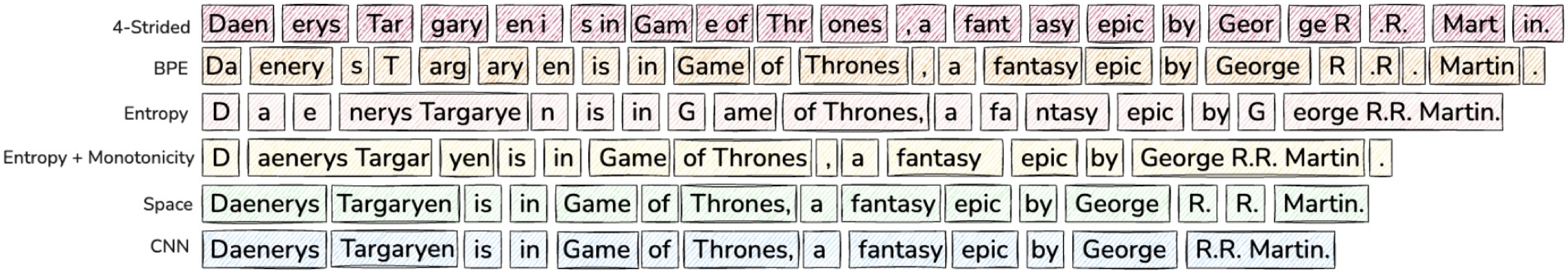
- Domain sensitivity: Tokenization schemes may underperform on out-of-domain data.
- Multilingual inequity: Tokenization is less efficient for languages with complex scripts.
- Poor handling of noisy inputs (e.g., typos, formatting issues).

**Compute Allocation Challenges:**

- Equal compute for all tokens, regardless of complexity.
- Inefficient for simple predictions like common endings.

**Orthographic Knowledge Gap:**

Token-based models lack intrinsic understanding of sub-word and character-level patterns.


4-Strided Daen erys Tar gary en i s in Gam e of Thr ones , a fant asy epic by Geor ge R .R. Mart in.
BPE Da enery s T arg ary en is in Game of Thrones , a fantasy epic by George R .R . Martin .
Entropy D a e nerys Targarye n is in G ame of Thrones, a fa ntasy epic by G eorge R.R. Martin.
Entropy + Monotonicity D aenerys Targar yen is in Game of Thrones , a fantasy epic by George R.R. Martin .
Space Daenerys Targaryen is in Game of Thrones, a fantasy epic by George R. R. Martin.
CNN Daenerys Targaryen is in Game of Thrones, a fantasy epic by George R.R. Martin.

# Byte-Level Tokenizer Model (BLT)

In 2024, Meta released the BLT model, addressing these issues

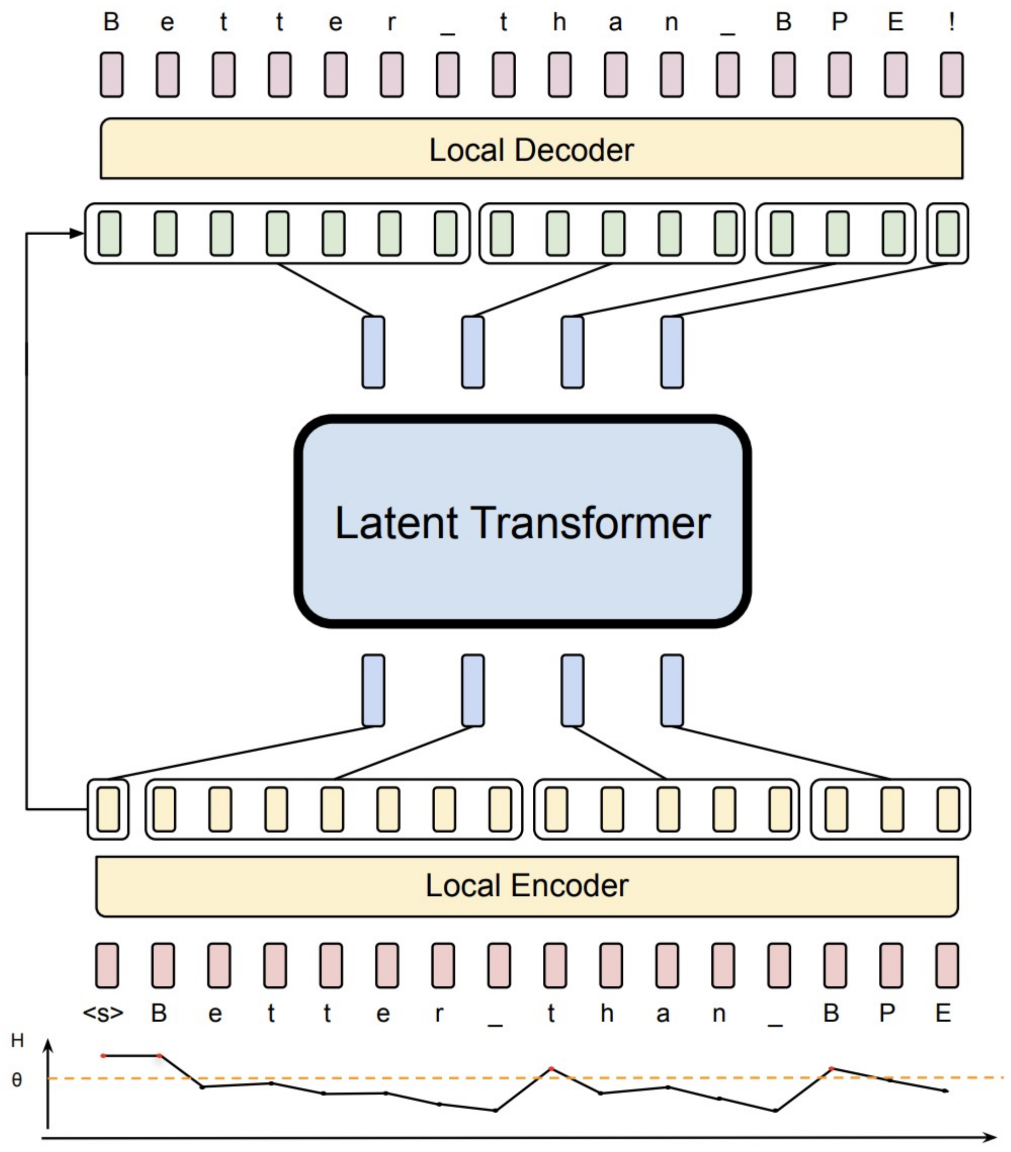
**Dynamic Byte-Level Patching:**
Segments raw bytes into entropy-based “patches” of variable sizes.
Allocates compute where needed, based on data complexity.

**Model Design:**

1. Local Encoder: Encodes raw bytes into patches.
2. Latent Transformer: Processes patch representations.
3. Local Decoder: Decodes patches back into bytes.

**Advantages Over Tokenization:**

- No fixed vocabulary or heuristic compression.
- Robust to noisy inputs and better character-level understanding.
- More efficient scaling by dynamically adjusting patch size and model capacity.



B e t t e r _ t h a n _ B P E !
Local Decoder
Latent Transformer
Local Encoder
<s> B e t t e r _ t h a n _ B P E
H
θ

5. Small Byte-Level Transf. Makes **Next-Byte Prediction**

4. **Unpatching** to Byte Sequence via Cross-Attn

3. Large Latent Transformer **Predicts Next Patch**

2. Entropy-Based Grouping of Bytes Into **Patches** via Cross-Attn

1. Byte-Level Small Transf. Encodes **Byte Stream**

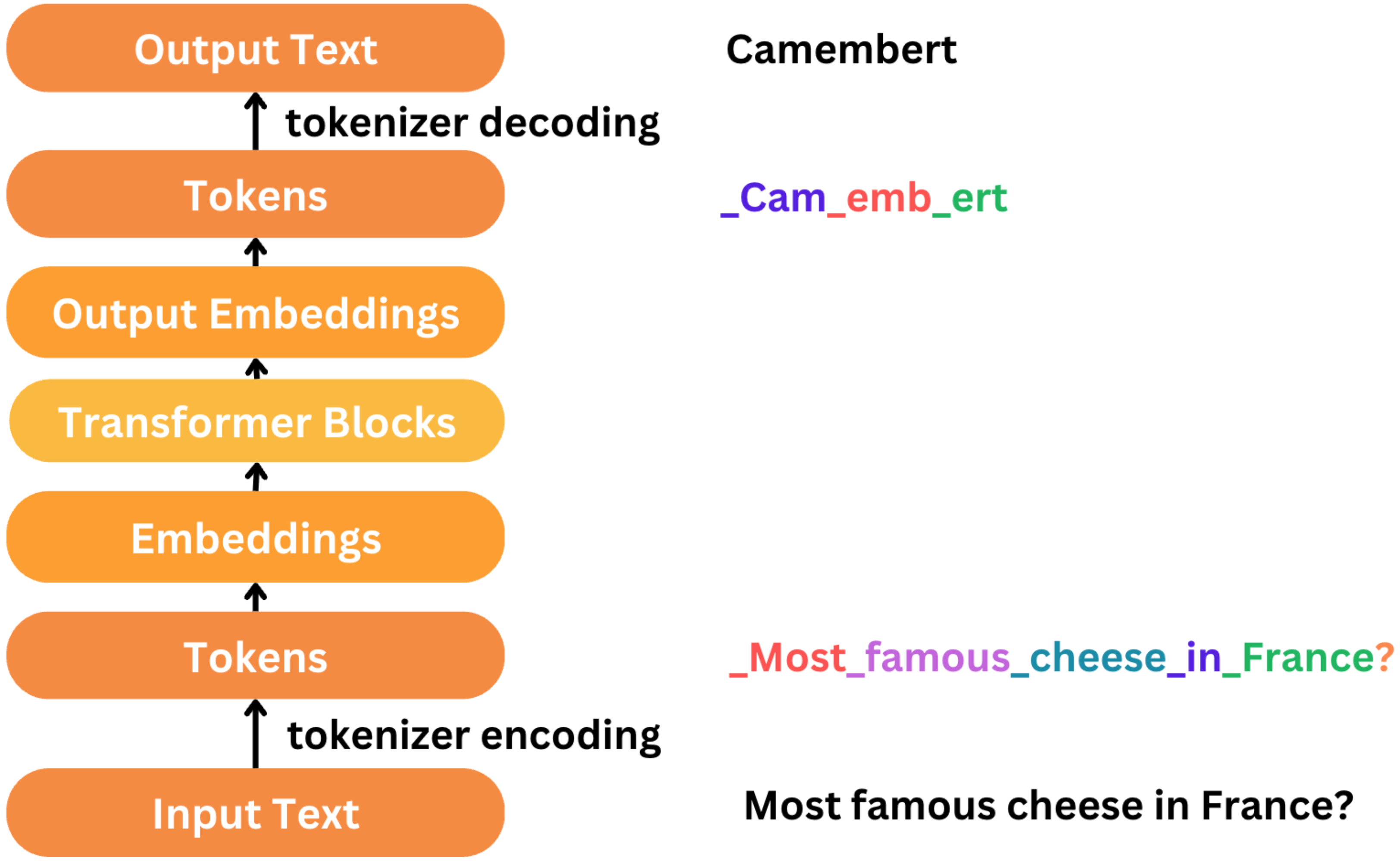
# Wrap Up

## Tokens and Tokenization

- Today we introduced the concepts of tokens and tokenization.
- We saw different approaches to tokenizing text and their relative trade-offs.
- Some of the common issues with modern large language models were identified as rooted in tokenization
- A new model, based on a token-free approach was also discussed, highlighting the need for ongoing research in this aspect of NLP.

-----------------------------------------------------------------------------

In the next class we'll look into how we convert these tokens into word vectors to codify meaning and allow these language models to perform better



NVIDIA | DARTMOUTH ENGINEERING

# Thank you!