Tokenization and Embedding in Large Language Models (LLMs)

Dr. Saikat Sarkar

Assistant Professor

Bangabasi College, University of Calcutta, Kolkata, India

to.saikatsarkar17@gmail.com

Contents

What is an LLM (Large Language Model)?

- A Large Language Model (LLM) is an advanced AI model
- Trained on vast amounts of text (books, websites, and other sources)
- To understand, generate, and reason with human language.

LLM: Applications (Typical)

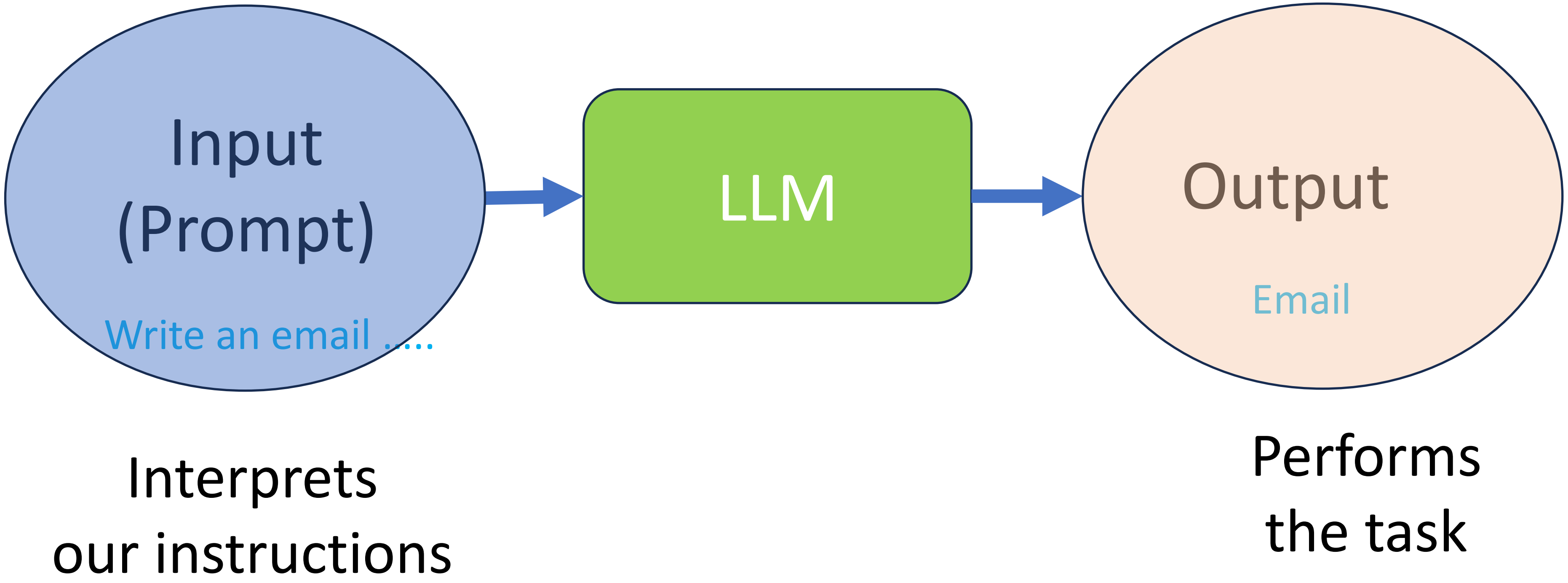
- Question answering
  - – Example: You can ask, “What is the capital of India?” and the model will reply “New Delhi”
- Writing essays, code
- Translating languages
- Generating creative text
  - – Example: The model can write a poem, story, or dialogue between two characters.

Big players

- ChatGPT: by OpenAI
- Big tech
  - Gemini: Google's version
  - Meta AI: Meta's version
  - Copilot: Microsoft's version
- Startups ---
  - Claude: Anthropic's version
  - Grok: xAI's version
  - Perplexity
- DeepSeek (Chinese Co.)
- Le Chat: Mistral's version (French Co.)

LLM Workflow

LLM Workflow



Criteria for Interpretation in LLMs

- Task Type Recognition:
  - Write an email
  - Write a story

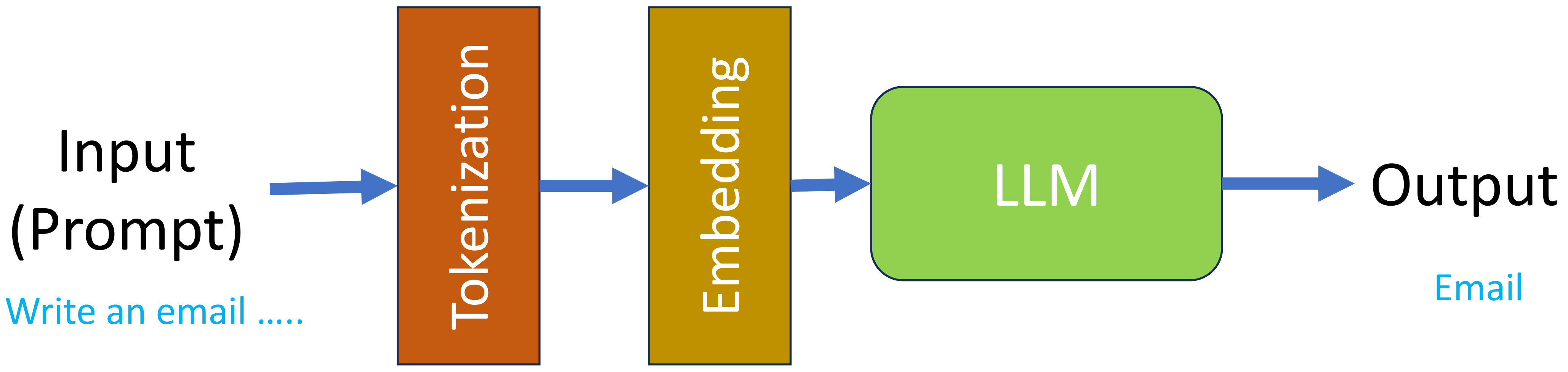
- Intent Detection:
  - Explain photosynthesis: Command
  - What is photosynthesis? Question

- Polarity and Sentiment Recognition:
  - This movie was amazing: positive
  - This movie was terrible: negative
- Context:
  - He sat on the bank: river context
  - He went to the bank: finance context

How LLMs Understand Text

- LLMs can’t understand words or sentences directly — they work with numbers.
- Before feeding text to the model, we need to convert text into numbers that the model can process.
- That’s where Tokenization and Embedding come in!
  - Tokenization – Split text into tokens.
  - Embedding – Convert tokens into numerical vectors.

LLM Workflow: Tokenization and Embedding



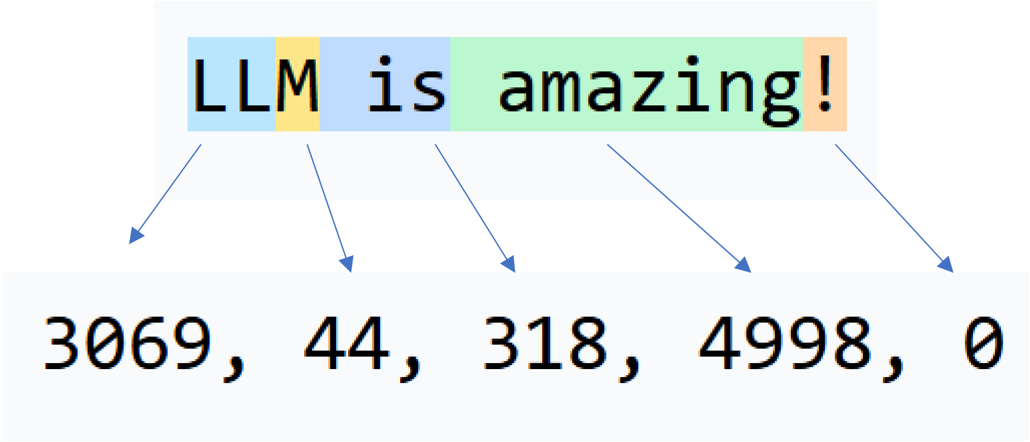
Split texts

Vector

What is Tokenization?

- **Tokenization** means breaking a sentence or text into smaller parts called **tokens**.

- These tokens can be:
  - Words
  - Subwords (parts of words)
  - Characters
  - Punctuation marks



https://tiktokenizer.vercel.app/?model=gpt2

Tokenization: Examples

- LLM vs llm
- LLM!

- 127 + 2895 * 164 = 474907
- 11+2*10=31
- 111+2*100=311

```
for i in range (10):
    print(i)
        if i> 5:
            exit(0)
```

Tokenization: Character Level

- Why not just use each character as a token?
  - “I am learning LLM”: 4 words, 17 characters

- LLMs process sequences token by token.
  - More tokens = longer sequences = more memory + compute time.

- Loss of semantic information
  - “unhappiness” → characters [u, n, h, a, p, p, i, n, e, s, s]
  - Each character gives almost no clue about meaning.

Tokenization: Word Level

- Why not just split every sentence by spaces and assign each word an ID?

- Vocabulary explosion (too many unique words)
  - “play”, “played”, “playing” → unrelated in word-level vocab.

- A word-level model cannot handle unseen words.
  - If the training data never saw “cyber-bioinformatics”, the model can’t represent it.
  - It will either replace it with [UNK] (unknown)
  - Or fail to interpret.

Modern LLMs use subword tokenization

Word	Sub-word tokens
play	["play"]
playing	["play", "ing"]
replay	["re", "play"]
replaying	["re", "play“, “ing”]

Byte Pair Encoding (BPE)

- Suppose the data to be encoded is "aaabdaaabac"
- Replace the most occurred pair "aa" with Z

```
ZabdZabac
Z=aa
```

- Repeat: replace "ab" with Y

```
ZYdZYac
Y=ab
Z=aa
```

- Repeat: replace "ZY" with X

```
XdXac
X=ZY
Y=ab
Z=aa
```

- Decompression is just the opposite steps

https://en.wikipedia.org/wiki/Byte-pair_encoding

WordPiece Tokenization

- Similar to the BPE
- Now, instead of just choosing the most frequent pair (like BPE), WordPiece uses a likelihood-based criterion.
- For each candidate pair (x, y), WordPiece estimates:

$$\mathrm{score}(x, y) = \frac{\mathrm{freq}(xy)}{\mathrm{freq}(x) \times \mathrm{freq}(y)}$$

- Merge the best pair based on the score

SentencePiece Tokenizer

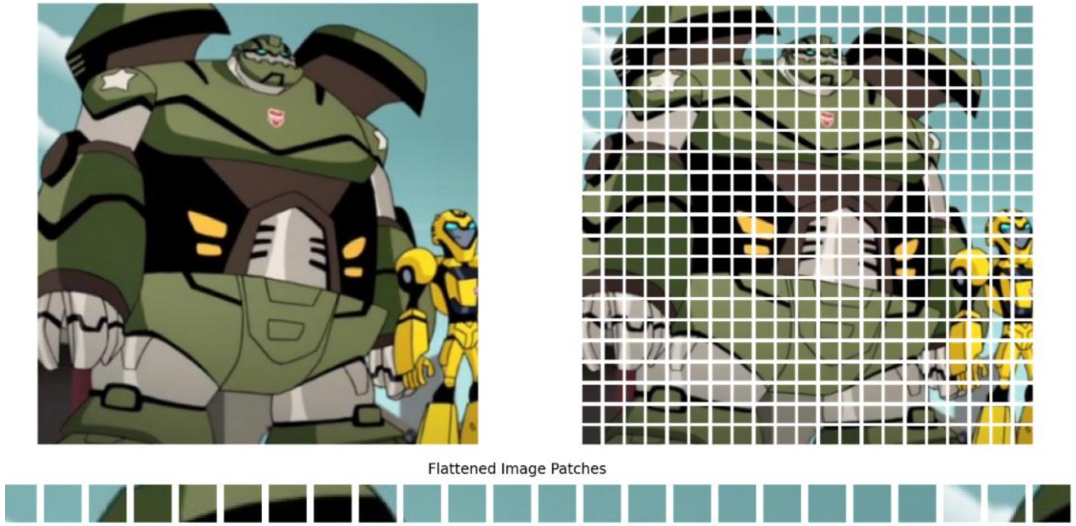
- BPE and WordPiece assume spaces = word boundaries
- That works well for English, but fails for:
  - Languages without spaces (e.g., Chinese, Japanese, Thai),
- Treats the entire text as a sequence of Unicode characters
- Keeps or replaces spaces with a special symbol, typically "▁"

  "I love learning" -> "▁I▁love▁learning"

  Could merge like "▁love" + "▁learn" → "▁love▁learn" (possible if common in corpus)

Tokenization: Non-textual Data

- Image: Sequence of patches



- Audio:

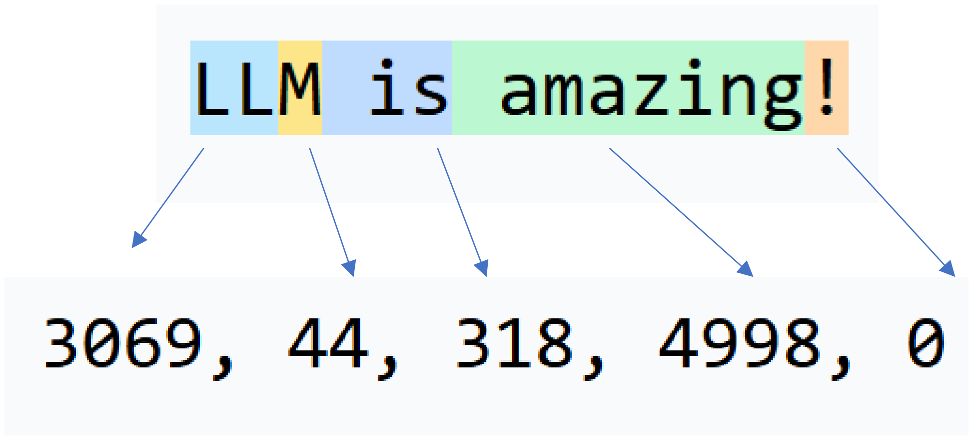
- Video

Questions?

From Tokens to Embeddings

Models don't use IDs directly — they use embeddings.

Embedding: a numerical vector that represents the meaning of a token.

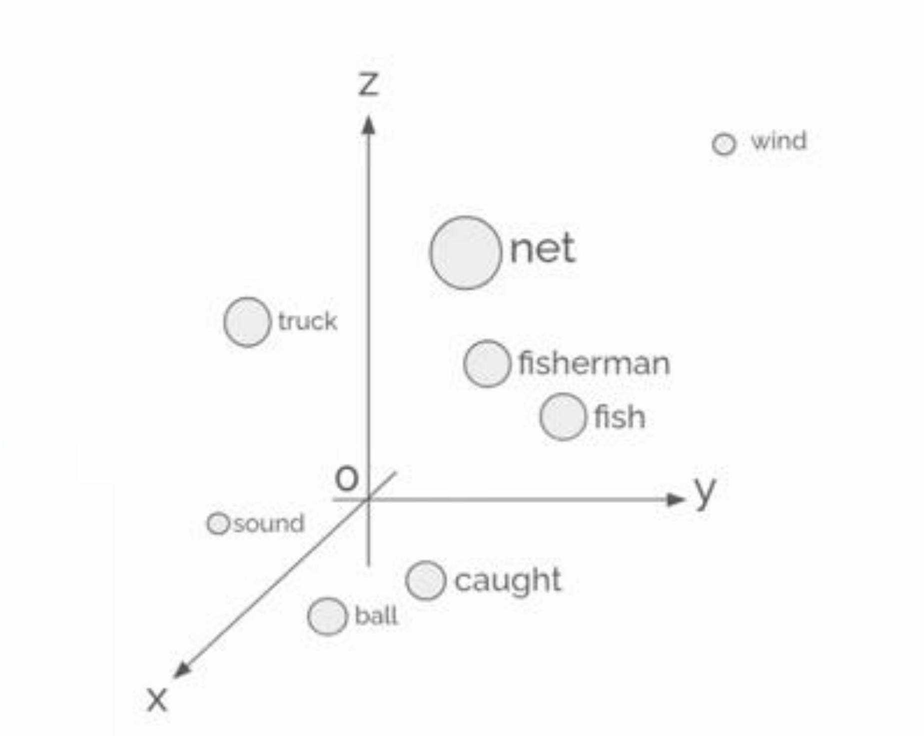


↓

Embedding Matrix:

[
  [0.19, -0.41, 0.88, ...],
  [0.73, -0.05, -0.32, ...],
  [-0.11, 0.25, 0.66, ...],
  ....
]

Visualizing Embedding Space



Word	Vector embedding (512 dimensions)
cat	[1.5, -0.4, 7.2, 19.6, 20.2,]
dog	[1.7, -0.3, 6.9, 19.1, 21.1, …]
fish	[-5.2, 3.1, 0.2, 8.1, 3.5, …]
fisherman	[-4.9, 3.6, 0.9, 7.8, 3.6, …]
triangle	[60.1, -60.3, 10, -12.3, 9.2, …]
PS4 Remote	[81.6, -72.1, 16, -20.2, 102, …]

Learning Embeddings

1. Start with random embeddings
2. Model predicts next word

   “The cat sat on the ___”

3. Compute error (loss)
4. Backpropagate and update embeddings
5. Similar words → similar embeddings

Token	Initial Embedding (simplified)
“cat”	[0.2, -0.4, 0.1]
“dog”	[-0.3, 0.5, -0.1]
“mat”	[0.8, -0.2, 0.3]

Positional Embedding

- Transformers process all tokens **in parallel**, not one after another.
- The model doesn't know the **order** of the words!

    The cat chased the dog

    The dog chased the cat

- We have to tell the model *where each word appears in the sequence.*

- A positional embedding is an additional vector added to each token's embedding that represents its position (index) in the sentence.

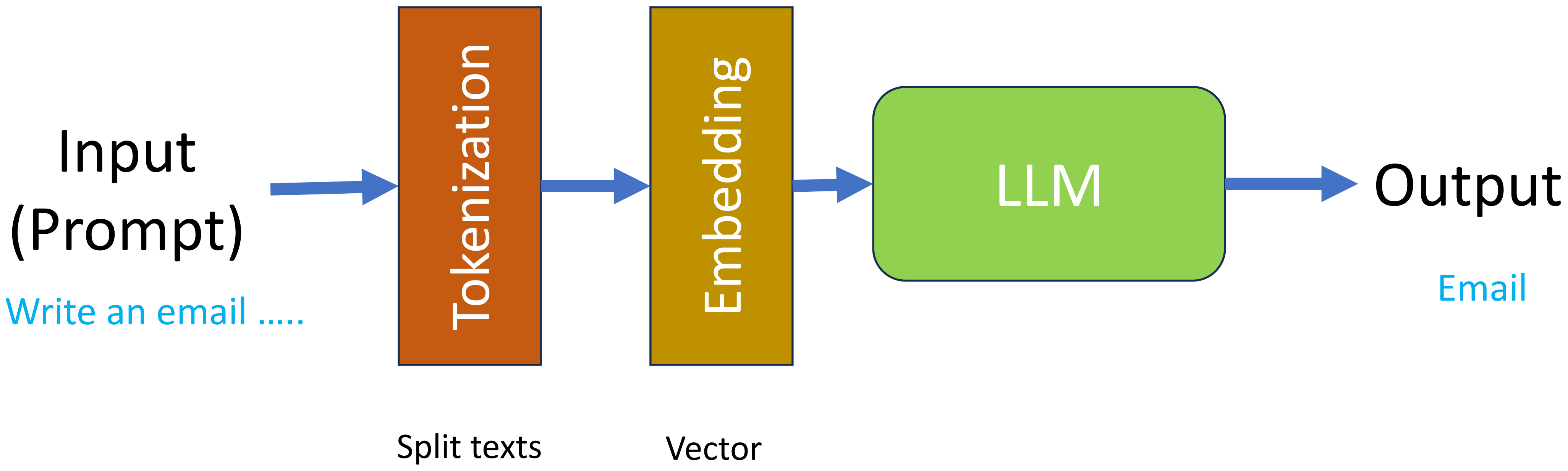
Positional Embedding

- Final input to the LLM = Token Embedding + Positional Embedding

Token	Token Embedding	Position	Positional Embedding	Final Input
“The”	[0.2, -0.1, 0.5]	1	[0.01, 0.99, 0.05]	[0.21, 0.89, 0.55]
“cat”	[0.8, -0.2, 0.3]	2	[0.02, 0.95, 0.10]	[0.82, 0.75, 0.40]
“sat”	[0.4, -0.4, 0.7]	3	[0.03, 0.90, 0.15]	[0.43, 0.50, 0.85]

Questions?

LLM Workflow: Tokenization and Embedding



Funny Things

- Counting Letters or Characters
  - How many letters are in the word *‘banana’*?”
- Counting Words Accurately
  - “Repeat the word ‘yes’ 50 times.”
  - The model predicts tokens, not actual word counts

References

- Intro to Large Language Models by Andrej Karpathy (https://www.youtube.com/watch?v=zjkBMFhNj_g)

- Introduction to large language models by

Google Cloud Tech (https://www.youtube.com/watch?v=zizonToFXDs)

- Transformers, explained: Understand the model behind ChatGPT

By Leon Petrou (https://www.youtube.com/watch?v=Pnd8bCJ4Z3A)

- Hands-On Large Language Models by Jay Alammar, Maarten Grootendorst
- Build a Large Language Model (From Scratch) by Sebastian Raschka

Thanks!!!