UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

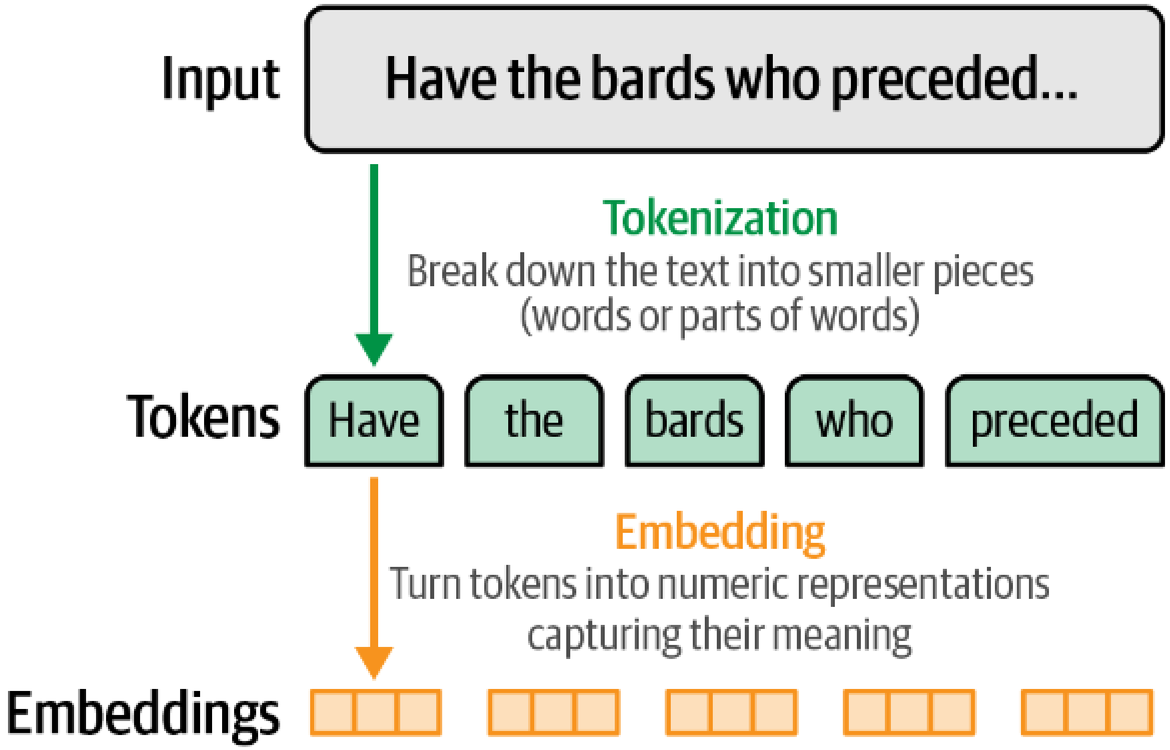
Artificial Intelligence

Lecture08 – NLP Tokenizers

Contenido

1. Tokenizers
2. BPE
3. Token embeddings

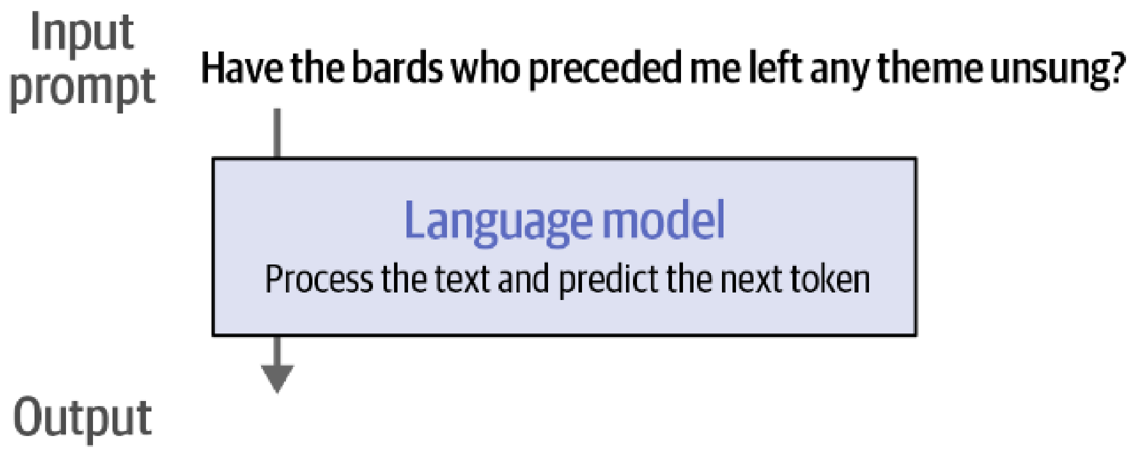
Embeddings


Input
Have the bards who preceded...
Tokenization
Break down the text into smaller pieces
(words or parts of words)
Tokens
Have
the
bards
who
preceded
Embedding
Turn tokens into numeric representations
capturing their meaning
Embeddings

LLM Tokenization

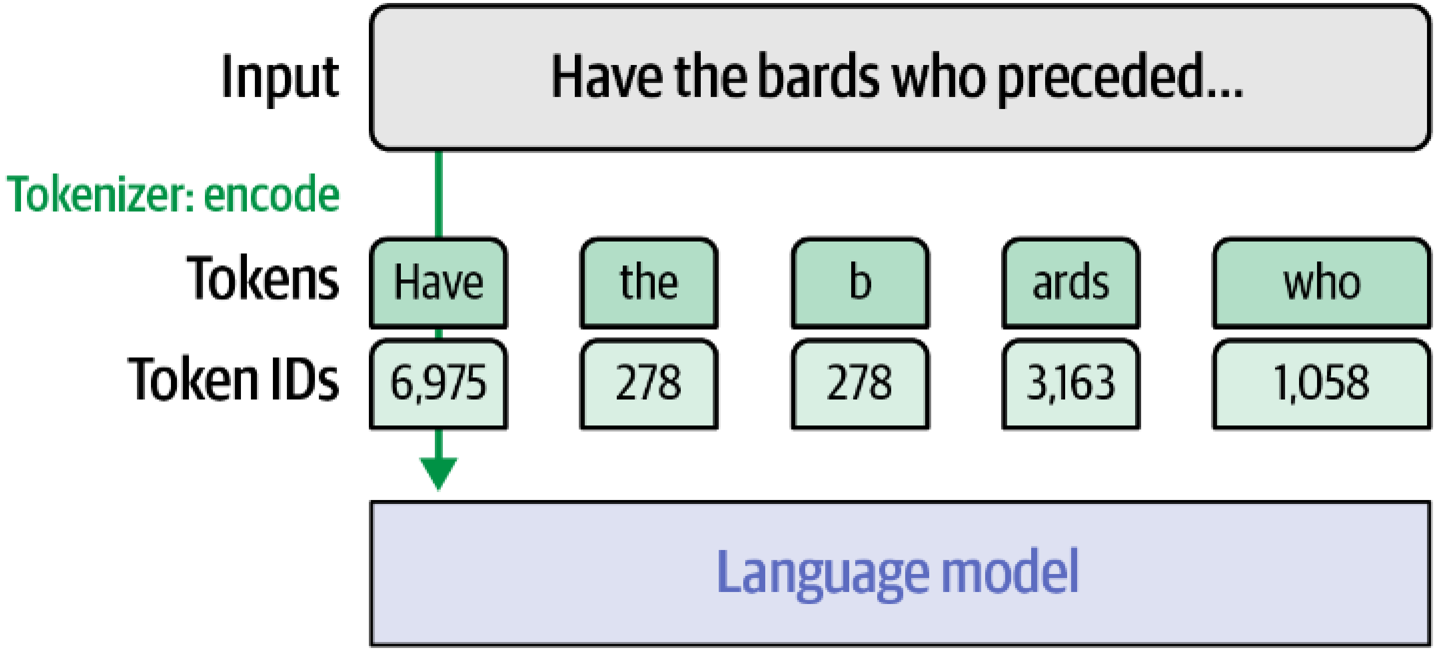
Universidad EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

How tokenizers prepare the inputs to the Language Model

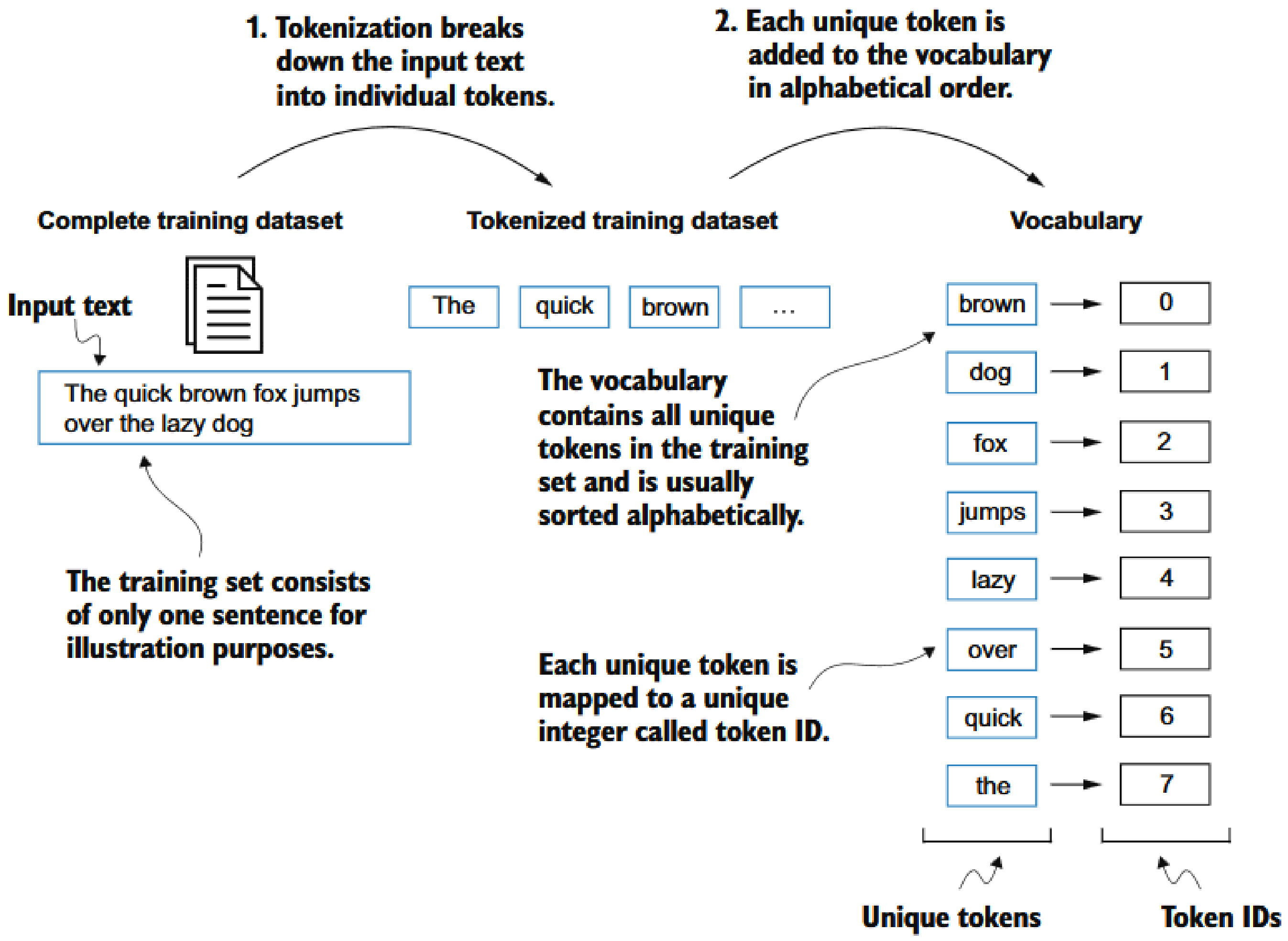


https://platform.openai.com/tokenizer

LLM Tokenization



LLM Tokenization

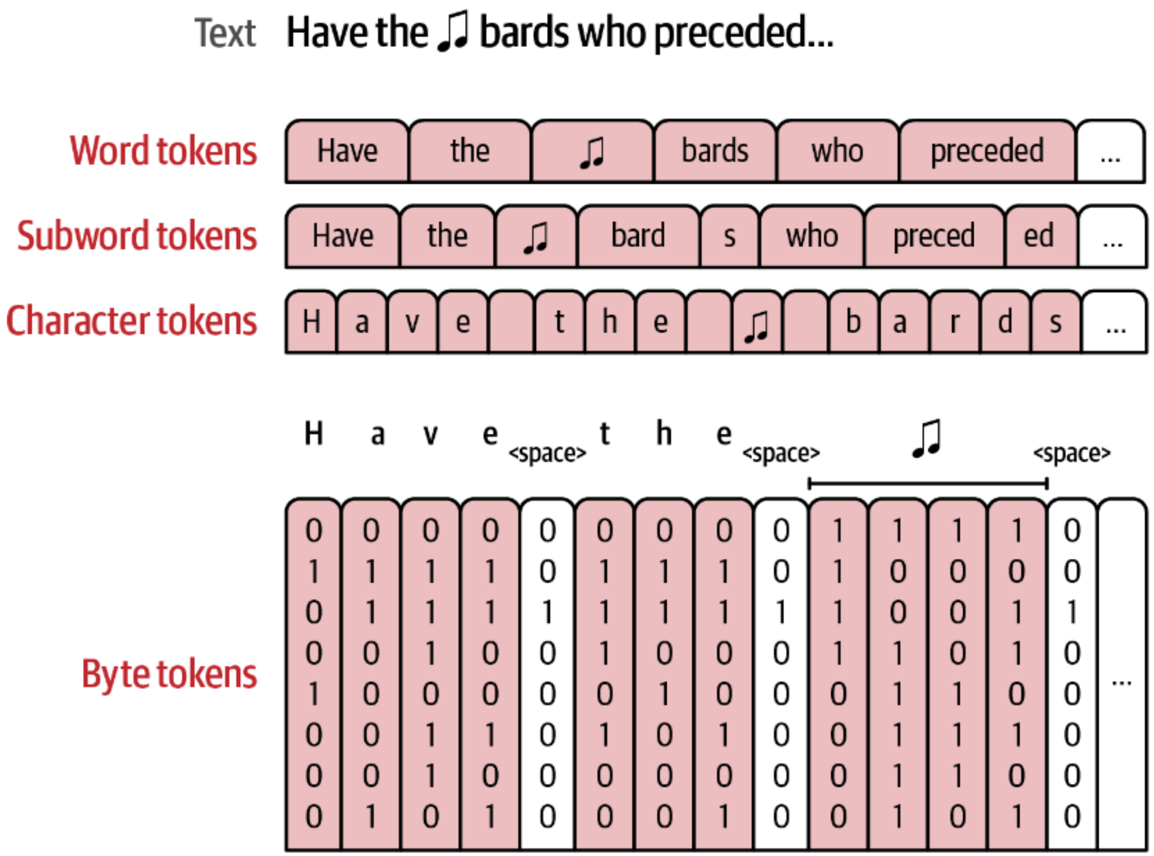


LLM Tokenization

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

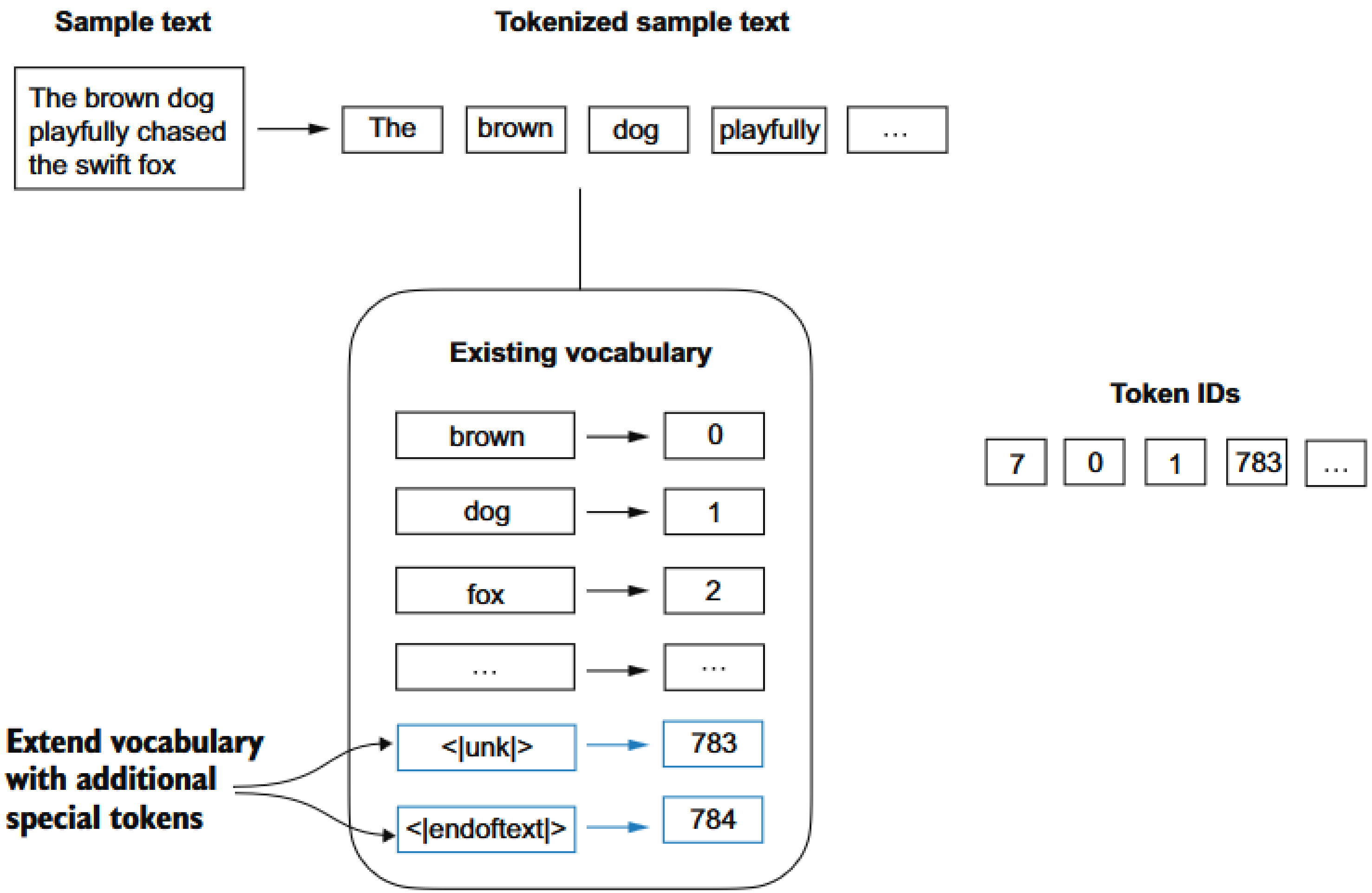
This is how the tokenization broke down the input token:

1. Some tokens are complete words (e.g., Write, an, email).
2. Some tokens are parts of words (e.g., apolog, izing, trag, ic).
3. Punctuation characters are their own token.

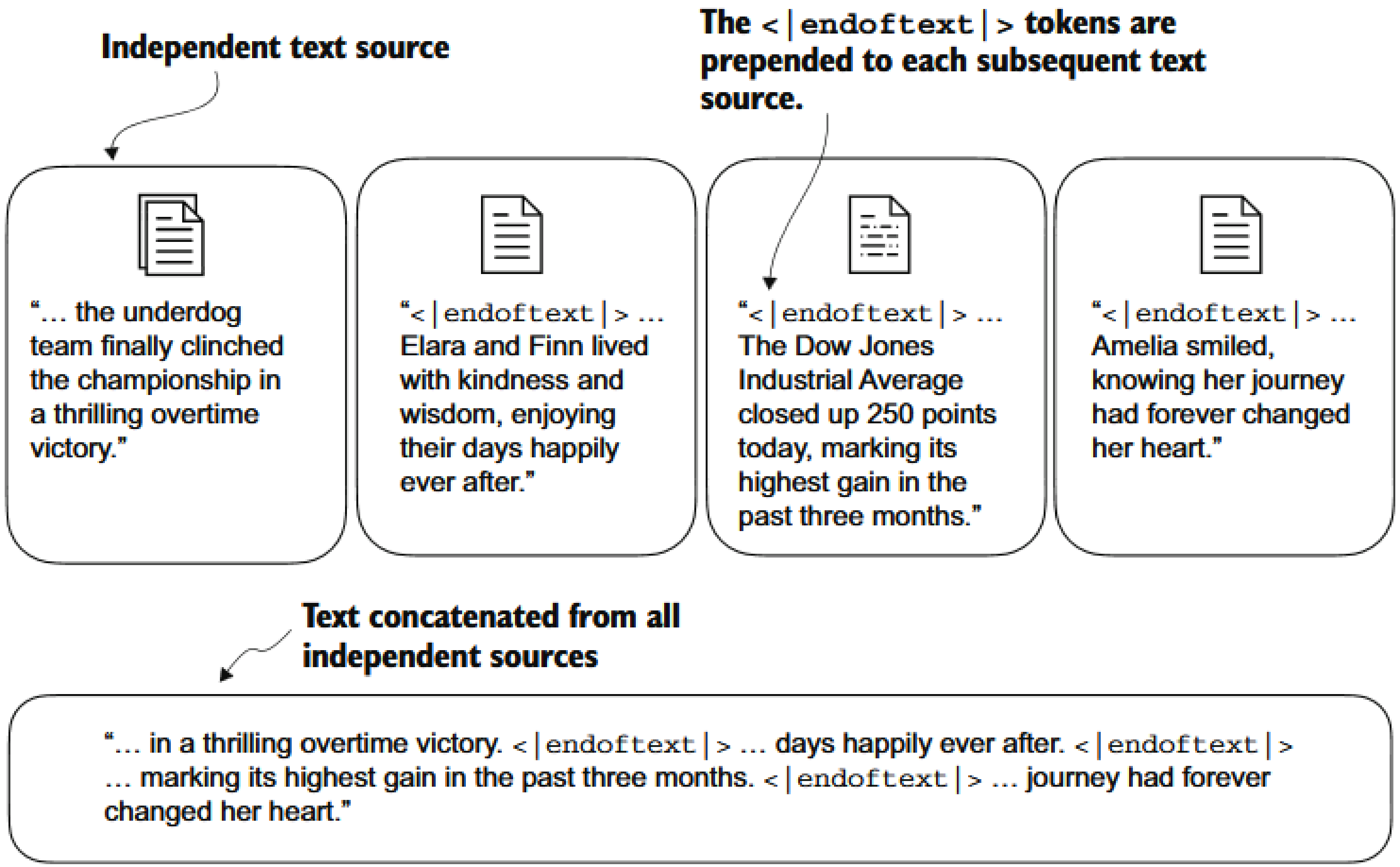


LLM Tokenization

Universidad EAFIT
Escuela de Ciencias Aplicadas e Ingeniería



LLM Tokenization



Byte Pair Encoding - BPE

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

GPT-2

Tokenization method: Byte pair encoding (BPE), introduced in "Neural machine translation of rare words with subword units".

Vocabulary size: 50,257

Special tokens: <|endoftext|>

English and CAP ITAL IZ ATION

� � � � � �

show _ t ok ens False None el if == >= else : two tabs :" " Three tabs : " "

12 . 0 * 50 = 600

Neural Machine Translation of Rare Words with Subword Units

Rico Sennrich and **Barry Haddow** and **Alexandra Birch**
School of Informatics, University of Edinburgh
{rico.sennrich,a.birch}@ed.ac.uk, bhaddow@inf.ed.ac.uk

GPT-4

- The GPT-4 tokenizer represents the four spaces as a single token. In fact, it has a specific token for every sequence of whitespaces up to a list of 83 whitespaces.
- The Python keyword elif has its own token in GPT-4. Both this and the previous point stem from the model's focus on code in addition to natural language.

Tokenizer parameters

📌 **Vocabulary Size**

- How many tokens the tokenizer can use.
- Common choices: **30K, 50K, 100K+ tokens** (larger vocabularies becoming more common).
- Larger vocabularies reduce sequence length but increase memory usage.

📌 **Special Tokens**

- Extra tokens added to handle specific cases.
- Common examples:
  - **<s>** → Beginning of text token.
  - **</s>** → End of text token.
  - **<pad>** → Padding token for sequence alignment.
  - **<unk>** → Unknown token for unseen words.
  - **<cls>** → Classification token (e.g., for BERT).
  - **<mask>** → Masking token for masked language models.

- **Custom tokens** can be added for domain-specific models (e.g., Galactica’s <work> and [START_REF]).

Tokenizer parameters

Universidad EAFIT — Escuela de Ciencias Aplicadas e Ingeniería

📌 **Capitalization Handling**

- Should the tokenizer preserve **capitalization**, or convert everything to **lowercase**?
- **Pros of keeping capitalization:** Maintains important information (e.g., "Apple" vs. "apple").
- **Cons:** Increases vocabulary size (separate tokens for "Hello" and "hello").
- **Trade-off:** Some tokenizers store only lowercase forms, using case markers instead.

Byte Pair Encoding - BPE

Universidad EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

1. Identify frequent pairs

- In each iteration, scan the text to find the most commonly occurring pair of bytes (or characters)

2. Replace and record

- Replace that pair with a new placeholder ID (one not already in use, e.g., if we start with 0…255, the first placeholder would be 256)
- Record this mapping in a lookup table
- The size of the lookup table is a hyperparameter, also called "vocabulary size" (for GPT-2, that's 50,257)

3. Repeat until no gains

- Keep repeating steps 1 and 2, continually merging the most frequent pairs
- Stop when no further compression is possible (e.g., no pair occurs more than once)

Decompression (decoding)

- To restore the original text, reverse the process by substituting each ID with its corresponding pair, using the lookup table

Token ID	Byte Value	Character Representation
0	0x00	NULL (NUL)
1	0x01	Start of Heading (SOH)
...	...	...
32	0x20	Space ( )
65	0x41	'A'
97	0x61	'a'
128	0x80	Extended ASCII
...	...	...
255	0xFF	Extended ASCII

BPE Example

- Suppose we have the text (training dataset) "the cat in the hat" from which we want to build the vocabulary for a BPE tokenizer
- **Iteration 1**
  1. Identify frequent pairs

     In this text, th appears twice (at the beginning and before the second e)
  2. Replace and record

     Replace th with a new token ID that is not already in use, e.g., 256

     the new text is: <256>e cat in <256>e hat

     the new vocabulary is

```
0: ...
...
256: "th"
```

BPE Example

- **Iteration 2**

1. Identify frequent pairs

   In the text <256>e cat in <256>e hat, the pair <256>e appears twice
2. Replace and record

   replace <256>e with a new token ID that is not already in use, for example, 257.

   The new text is:

```
<257> cat in <257> hat
```

The updated vocabulary is:

```
0: ...
...
256: "th"
257: "<256>e"
```

BPE Example

- **Iteration 3**

  1. Identify frequent pairs

     In the text <257> cat in <257> hat, the pair <257> appears twice (once at the beginning and once before “hat”).

  2. Replace and record

     Replace <257> with a new token ID that is not already in use, for example, 258.

     The new text is:

```
<258>cat in <258>hat
```

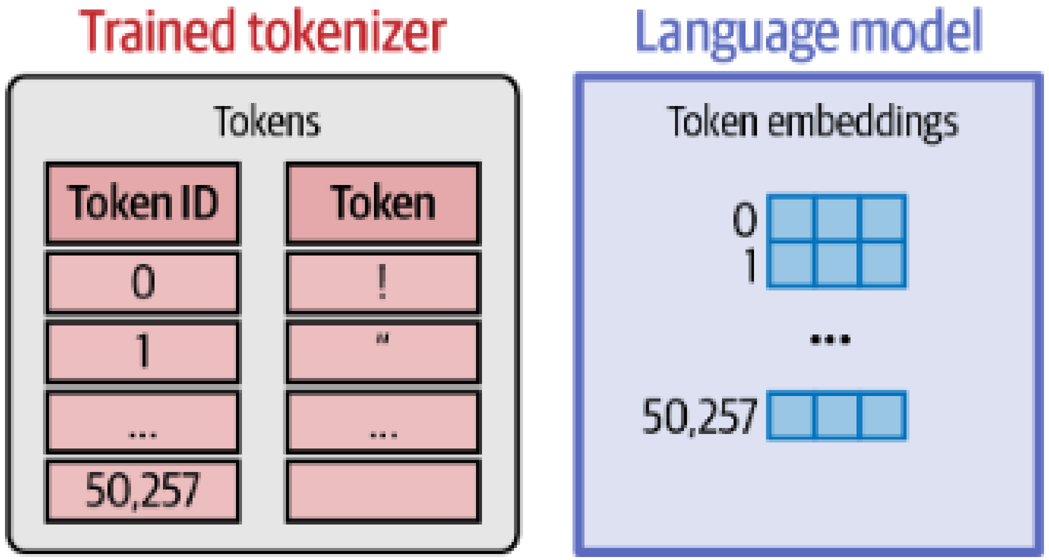
The updated vocabulary is:

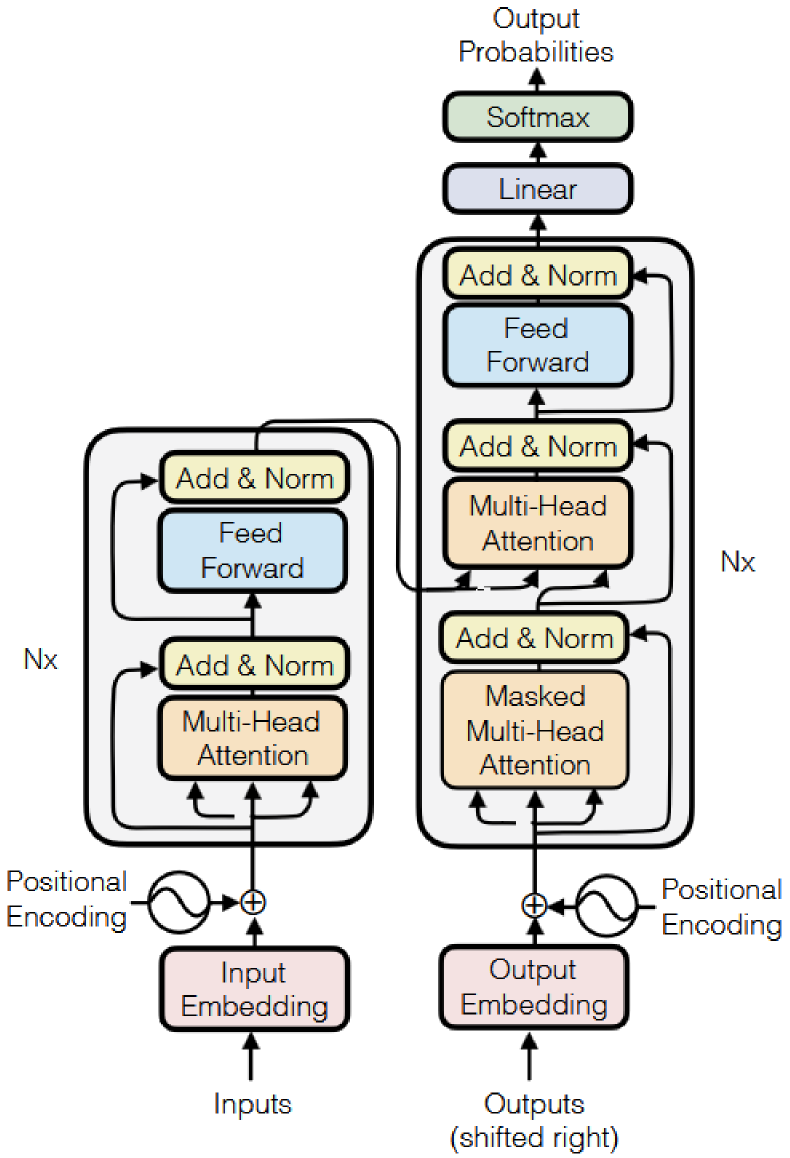
```
0: ...
...
256: "th"
257: "<256>e"
258: "<257> "
```

BPE Example

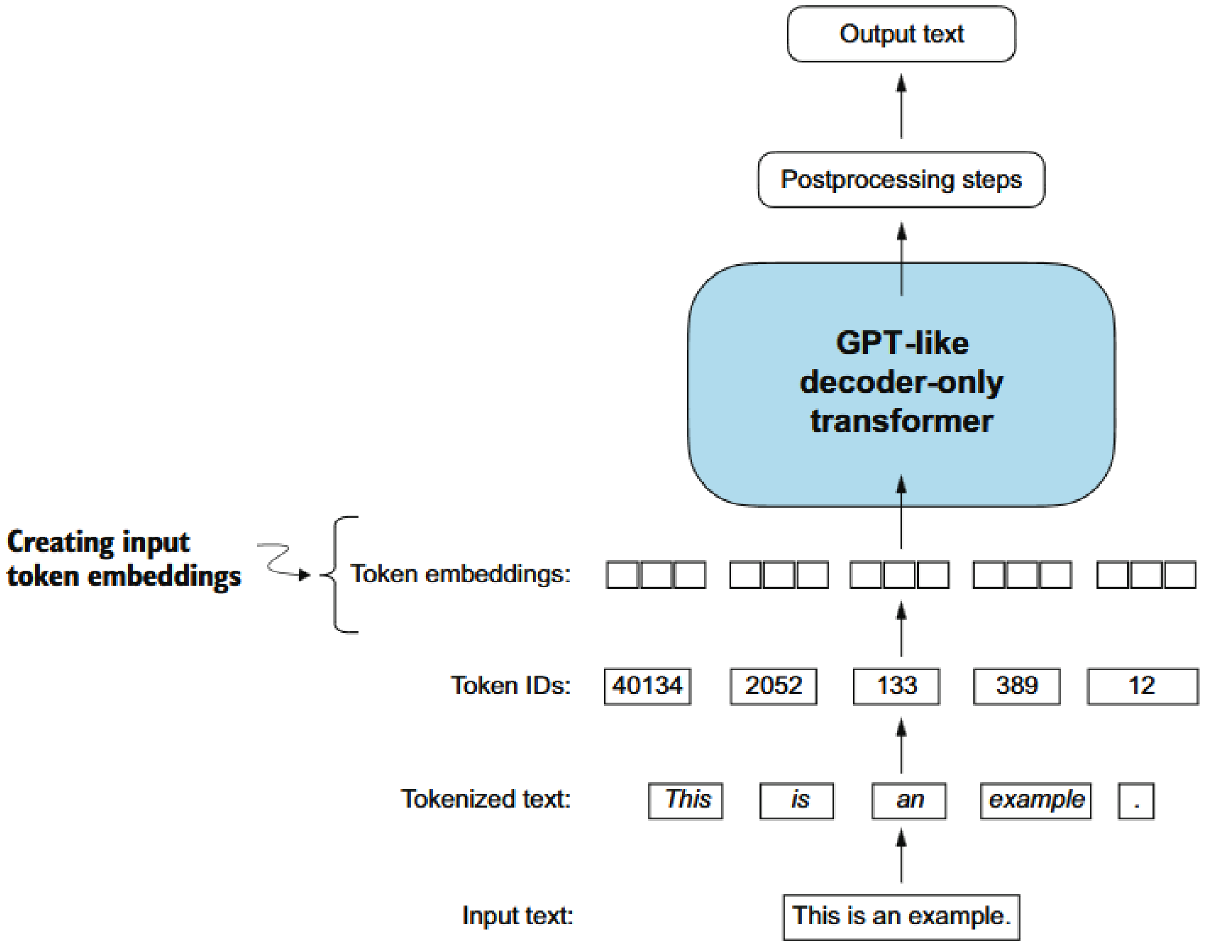
- To restore the original text, we reverse the process by substituting each token ID with its corresponding pair in the reverse order they were introduced
- Start with the final compressed text: <258>cat in <258>hat
- Substitute <258> → <257> : <257> cat in <257> hat
- Substitute <257> → <256>e: <256>e cat in <256>e hat
- Substitute <256> → “th”: the cat in the hat

Summary


Trained tokenizer
Tokens
Token ID
Token
0
!
1
"
...
...
50,257
Language model
Token embeddings
0
1
...
50,257


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Input Embedding
Inputs
Positional Encoding
Output Embedding
Outputs (shifted right)

Creating token embeddings



Creating token embeddings

Embedding layers: An efficient way of performing matrix multiplication when working with one-hot encoded vectors.

```python
import torch

torch.manual_seed(123);
```

```python
idx = torch.tensor([2, 3, 1]) # 3 training examples

num_idx = max(idx)+1
out_dim = 5
```

Input dimension of a one-hot encoded vector is the number of indices (the highest index + 1)

Suppose we want embeddings of size 5

Creating token embeddings

Universidad EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

```
import torch

torch.manual_seed(123);
```

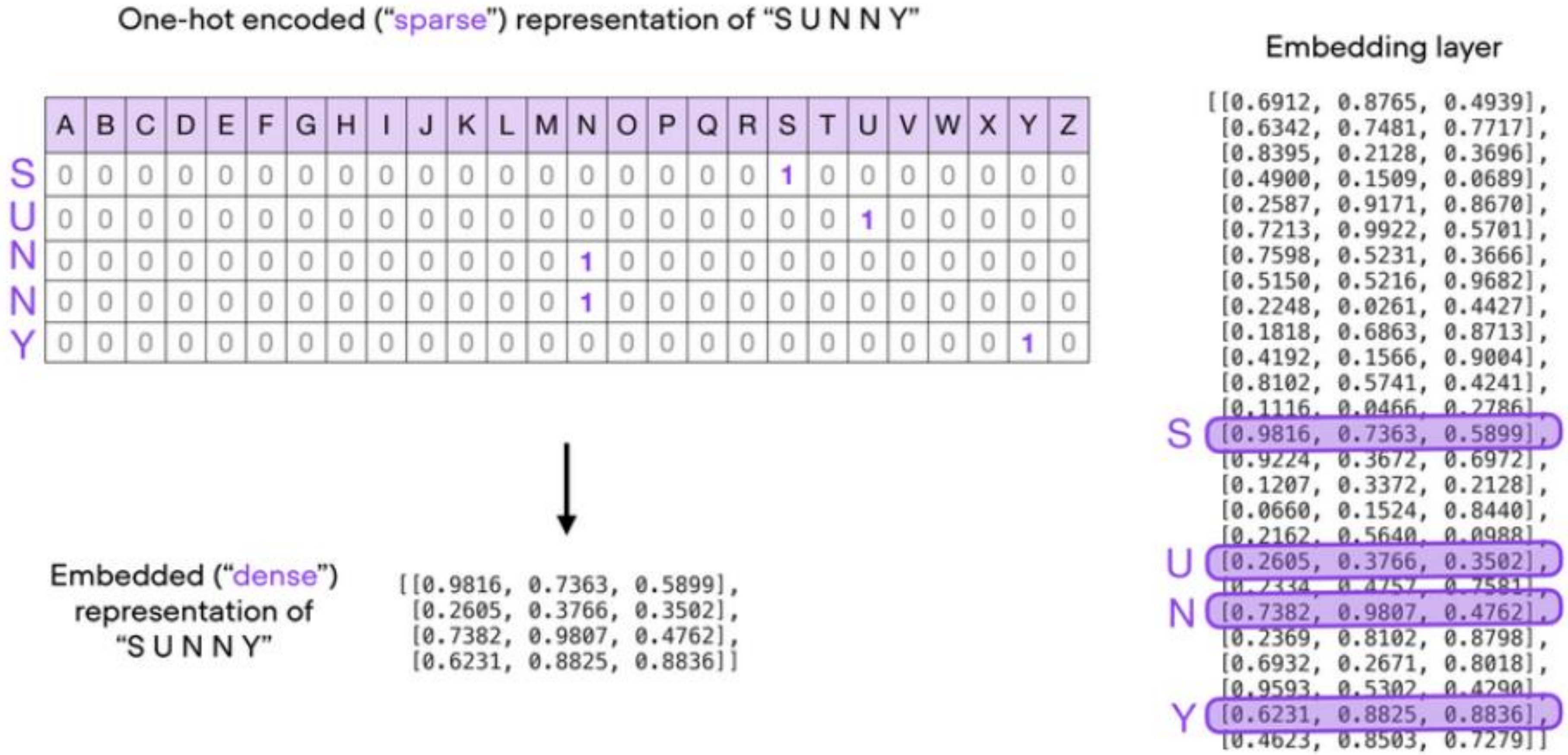
```
idx = torch.tensor([2, 3, 1]) # 3 training examples

num_idx = max(idx)+1
out_dim = 5

embedding = torch.nn.Embedding(num_idx, out_dim)
embedding(idx)
```

```
tensor([[ 0.6957, -1.8061, -1.1589,  0.3255, -0.6315],
        [-2.8400, -0.7849, -1.4096, -0.4076,  0.7953],
        [ 1.3010,  1.2753, -0.2010, -0.1606, -0.4015]],
       grad_fn=<EmbeddingBackward0>)
```

Each training example has 5 feature values

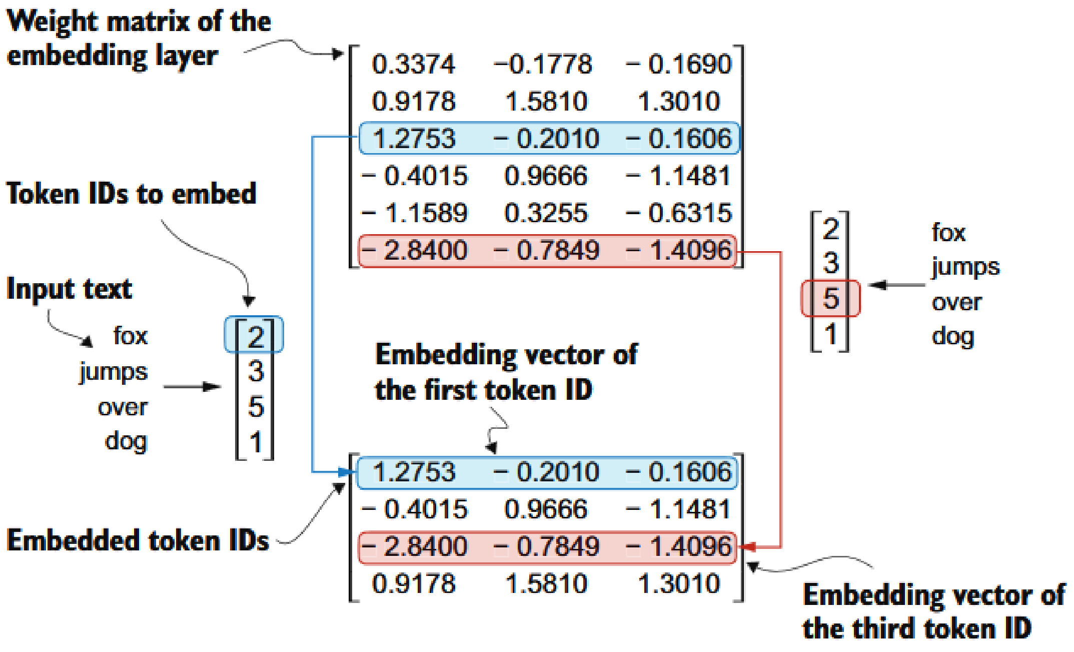
Creating token embeddings

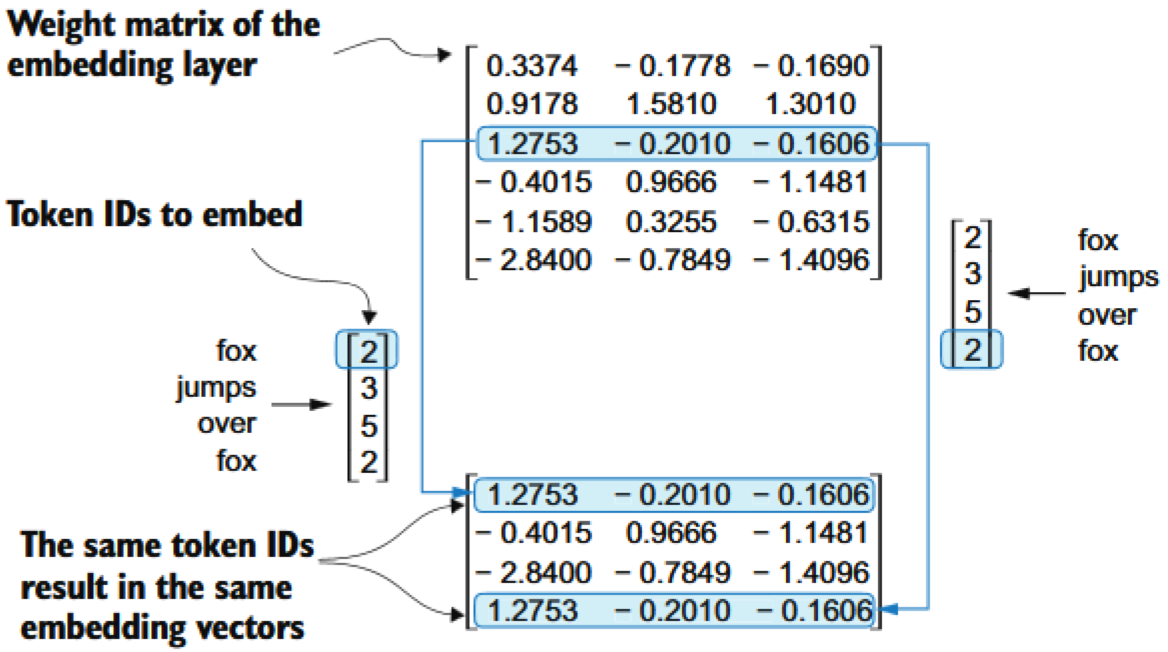
UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería



Creating token embeddings

Universidad EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

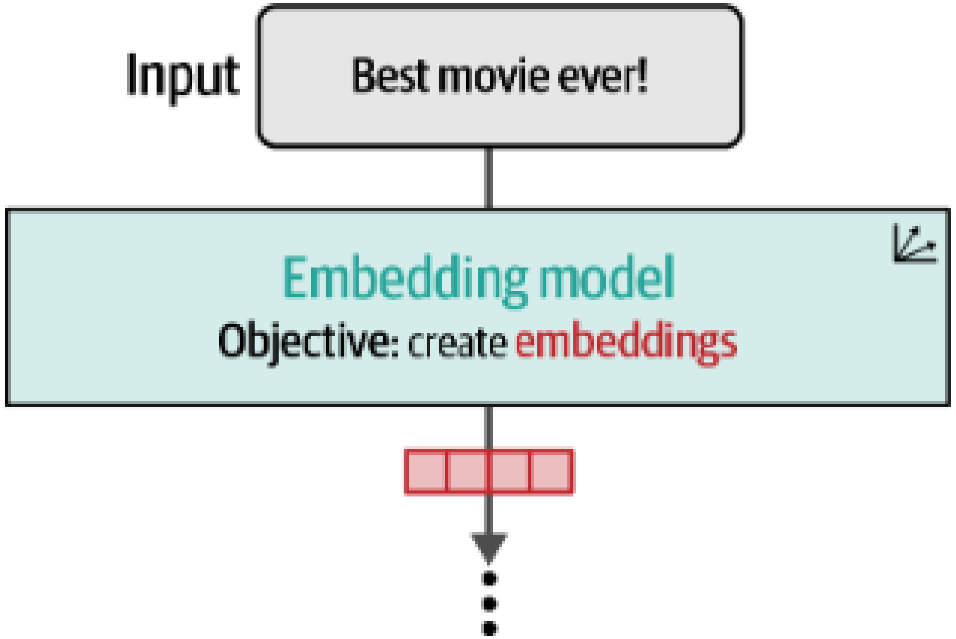




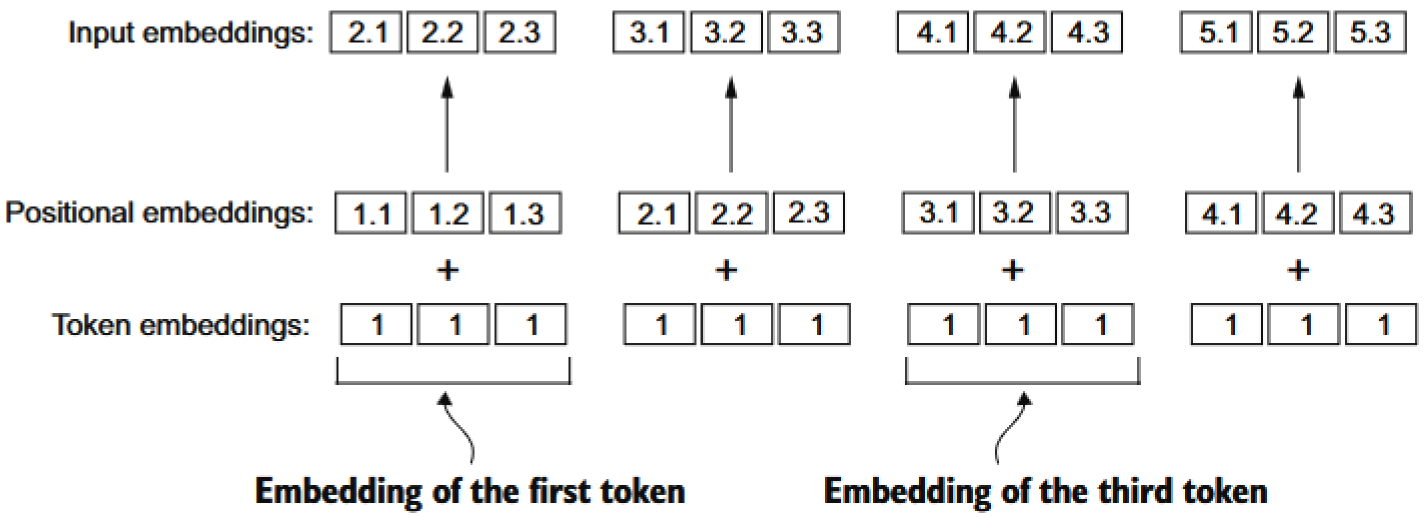
Sentence embeddings

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

Task: Aggregate token embeddings in a sentence to form a **unique sentence representation.**

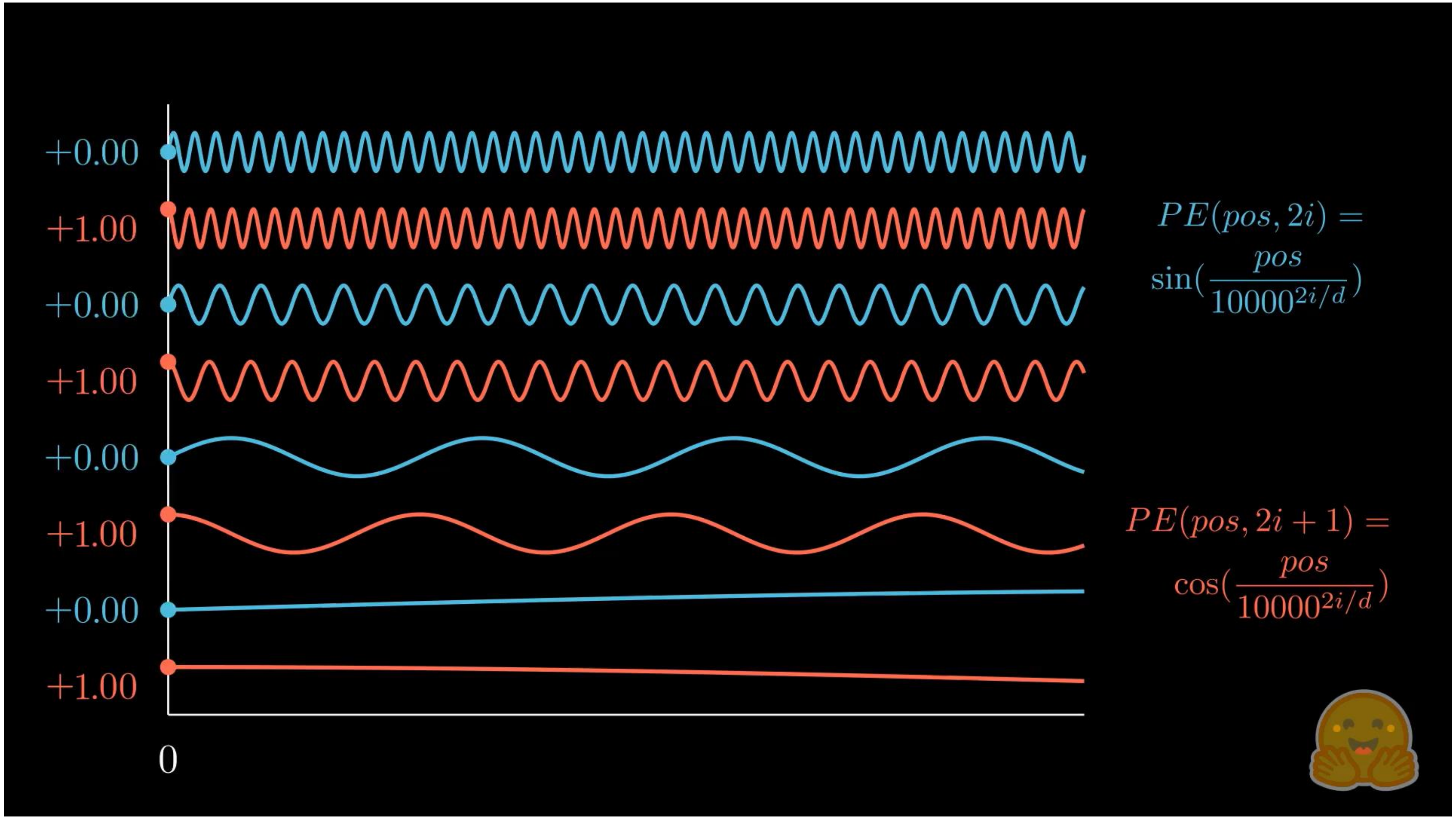


Encoding word positions



OpenAI: Positional embeddings are learnable parameters.
Llama, DeepSeek: Rotational Positional Embeddings – RoPE
Bert: Sinusoidal Positional Embeddings

Sinusoidal positional embeddings



$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

Rotary Positional Embeddings (RoPE)

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

Fundamental idea: we want the dot product of embeddings to result in a function of relative position:

$$f_q(\mathbf{x}_m, m) \cdot f_k(\mathbf{x}_n, n) = g(\mathbf{x}_m, \mathbf{x}_n, m-n)$$

ROFORMER: ENHANCED TRANSFORMER WITH ROTARY POSITION EMBEDDING

Jianlin Su, Zhuiyi Technology Co., Ltd., Shenzhen, bojonesu@wezhuiyi.com
Yu Lu, Zhuiyi Technology Co., Ltd., Shenzhen, julianlu@wezhuiyi.com
Shengfeng Pan, Zhuiyi Technology Co., Ltd., Shenzhen, nickpan@wezhuiyi.com
Ahmed Murtadha, Zhuiyi Technology Co., Ltd., Shenzhen, mengjiayi@wezhuiyi.com
Bo Wen, Zhuiyi Technology Co., Ltd., Shenzhen, brucewen@wezhuiyi.com
Yunfeng Liu, Zhuiyi Technology Co., Ltd., Shenzhen, glenliu@wezhuiyi.com

In summary, RoPE uses trigonometry to come up with a function that satisfies this property

$$R^d_{\Theta,m}\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{\frac{d}{2}} \\ \cos m\theta_{\frac{d}{2}} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_d \\ x_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{\frac{d}{2}} \\ \sin m\theta_{\frac{d}{2}} \end{pmatrix}$$

m is the token's position in the sequence.

$\Theta = (\theta_1, \theta_2, ..., \theta_{d/2})$ with:

$$\theta_i = 10000^{-2(i-1)/d}$$

Encoding word positions

Escuela de
Ciencias Aplicadas
e Ingeniería