

# Artificial Intelligence

## Lecture05 – NLP Tokenizers

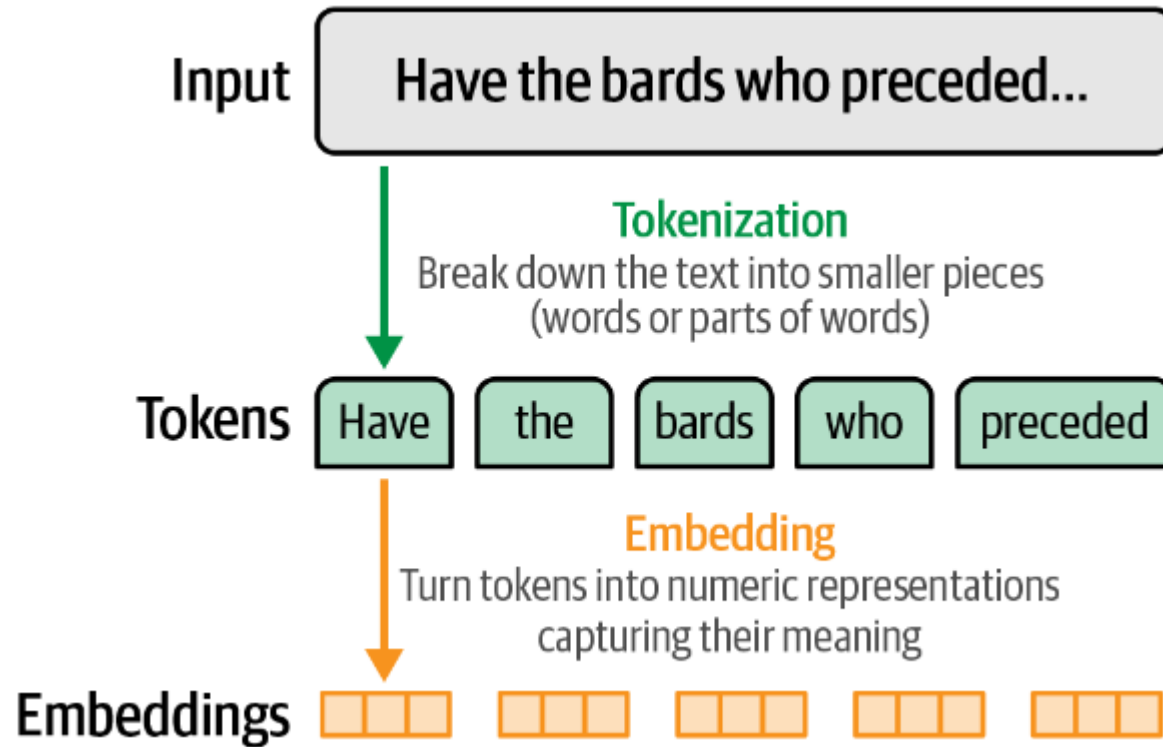


# Contenido

1. Tokenizers
2. BPE
3. Token embeddings

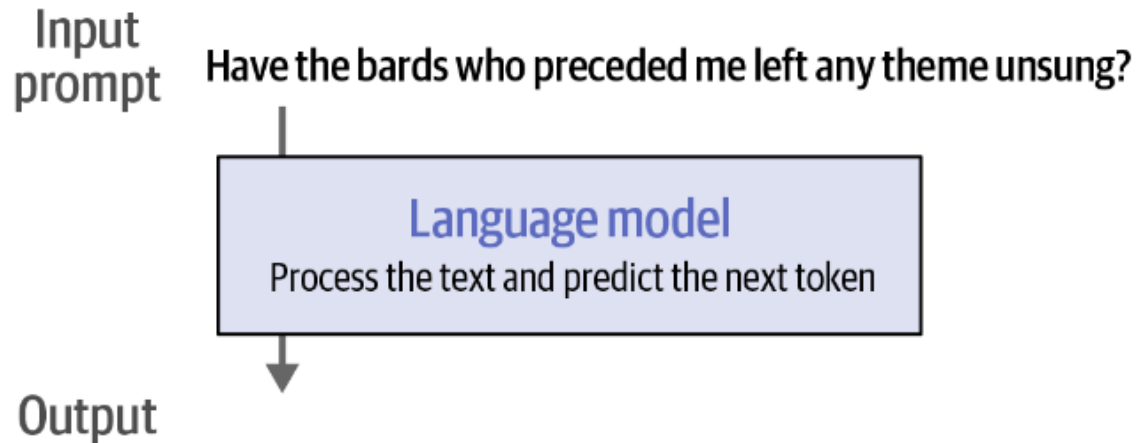


# Embeddings



# LLM Tokenization

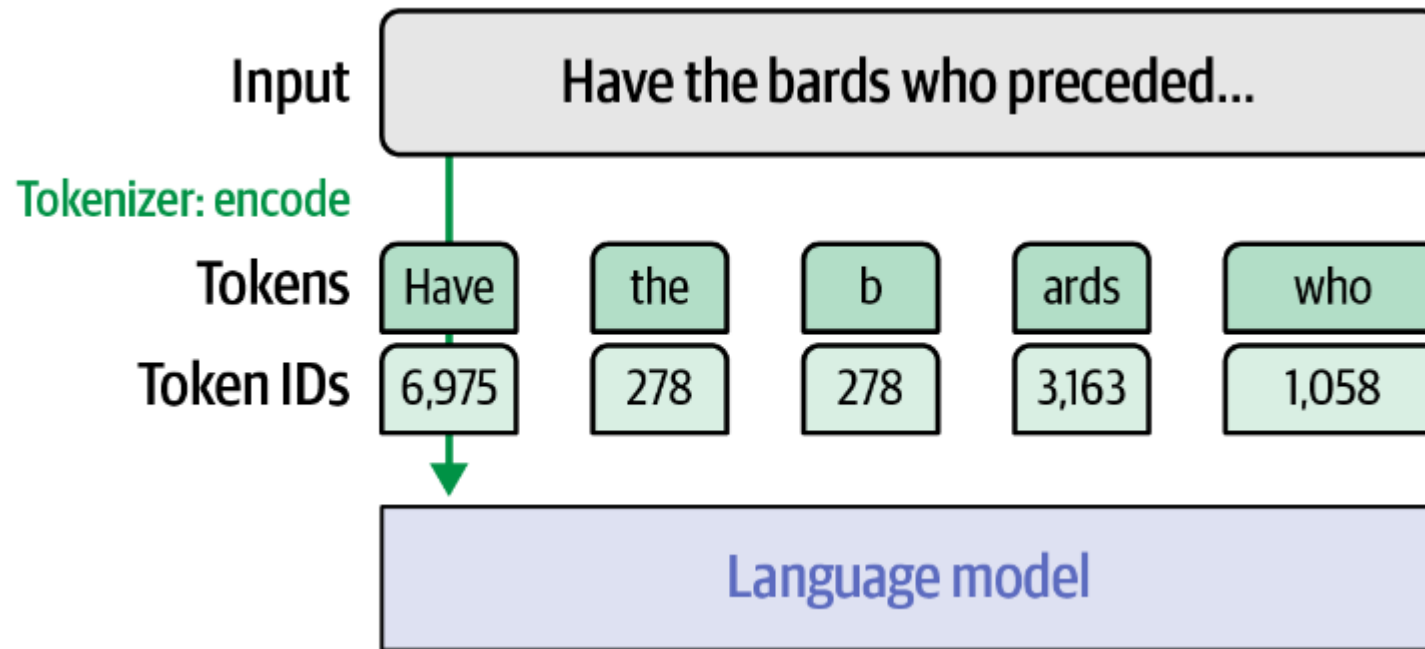
How tokenizers prepare the inputs to the Language Model



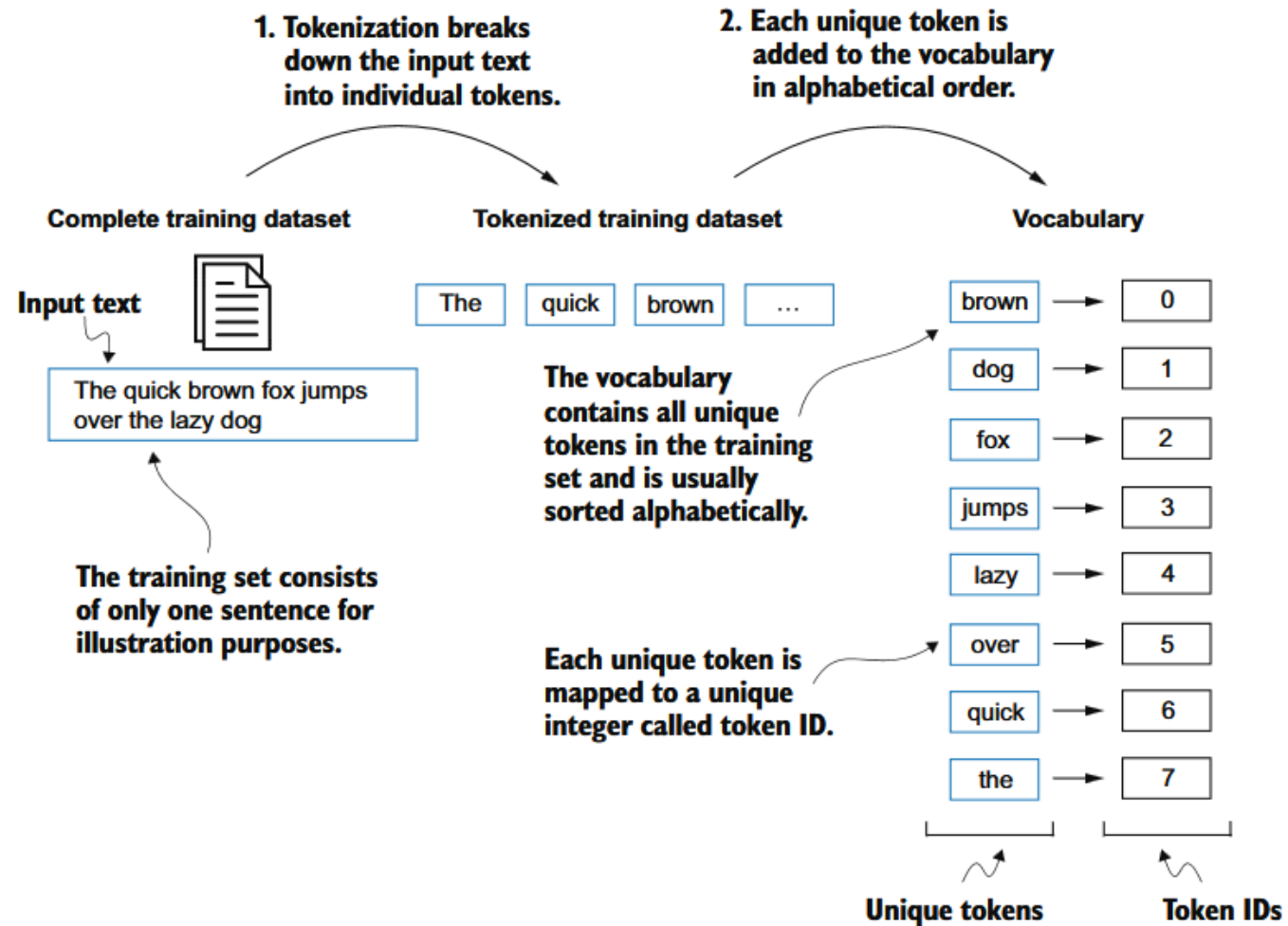
<https://platform.openai.com/tokenizer>



# LLM Tokenization



# LLM Tokenization



# LLM Tokenization

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.

Text Have the 🎵 bards who preceded...

Word tokens

Have	the	🎵	bards	who	preceded	...
------	-----	---	-------	-----	----------	-----

Subword tokens

Have	the	🎵	bard	s	who	preced	ed	...
------	-----	---	------	---	-----	--------	----	-----

Character tokens

H	a	v	e		t	h	e		🎵		b	a	r	d	s	...
---	---	---	---	--	---	---	---	--	---	--	---	---	---	---	---	-----

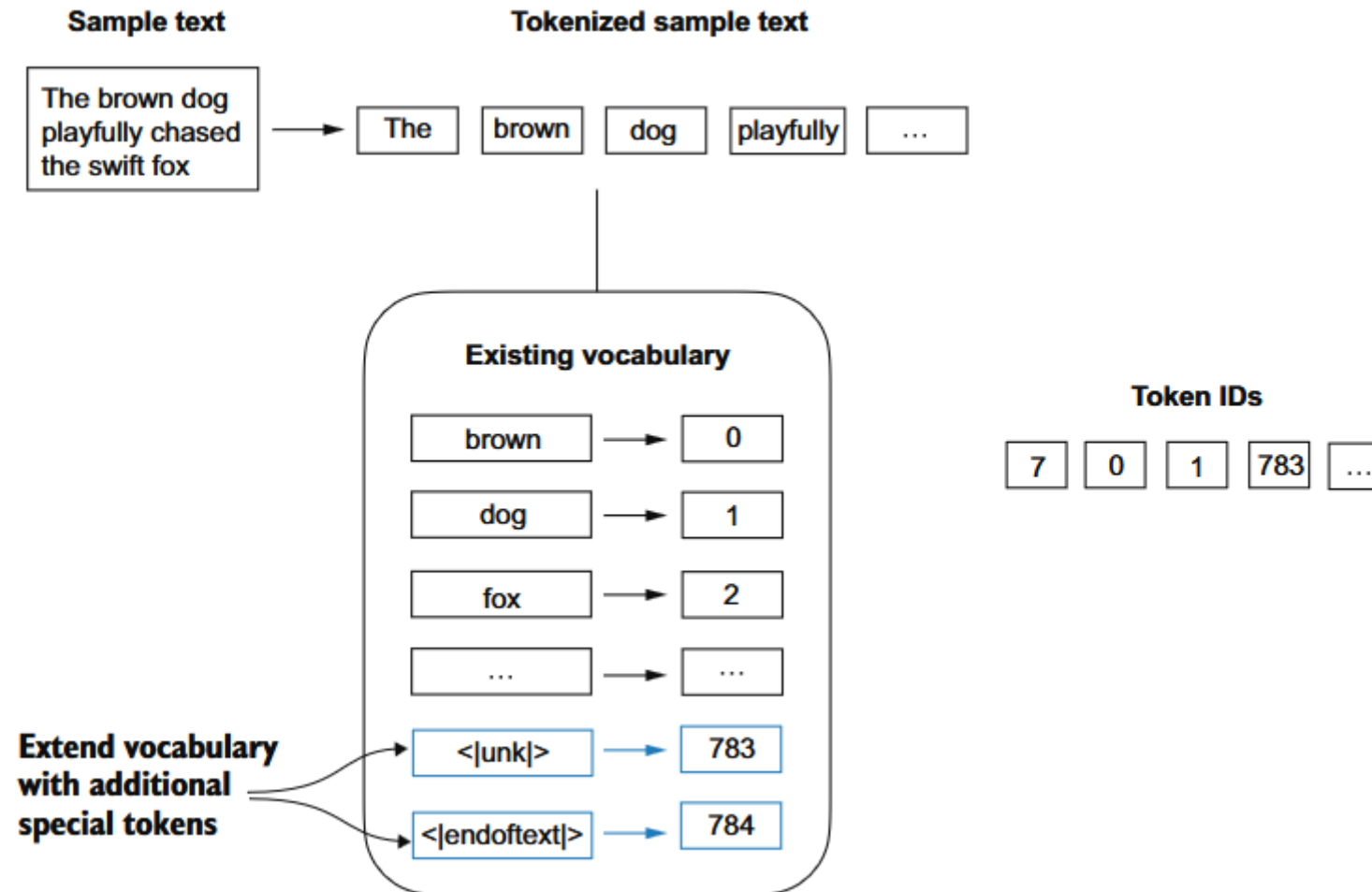
H a v e <space> t h e <space> 🎵 <space>

Byte tokens

0	0	0	0	0	0	0	0	0	1	1	1	1	0	
1	1	1	1	0	1	1	1	0	1	0	0	0	0	
0	1	1	1	1	1	1	1	1	1	0	0	1	1	
0	0	1	0	0	1	0	0	0	1	1	0	1	0	
1	0	0	0	0	0	1	0	0	0	1	1	0	0	...
0	0	1	1	0	1	0	1	0	0	1	1	1	0	
0	0	1	0	0	0	0	0	0	0	1	1	0	0	
0	1	0	1	0	0	0	1	0	0	1	0	1	0	

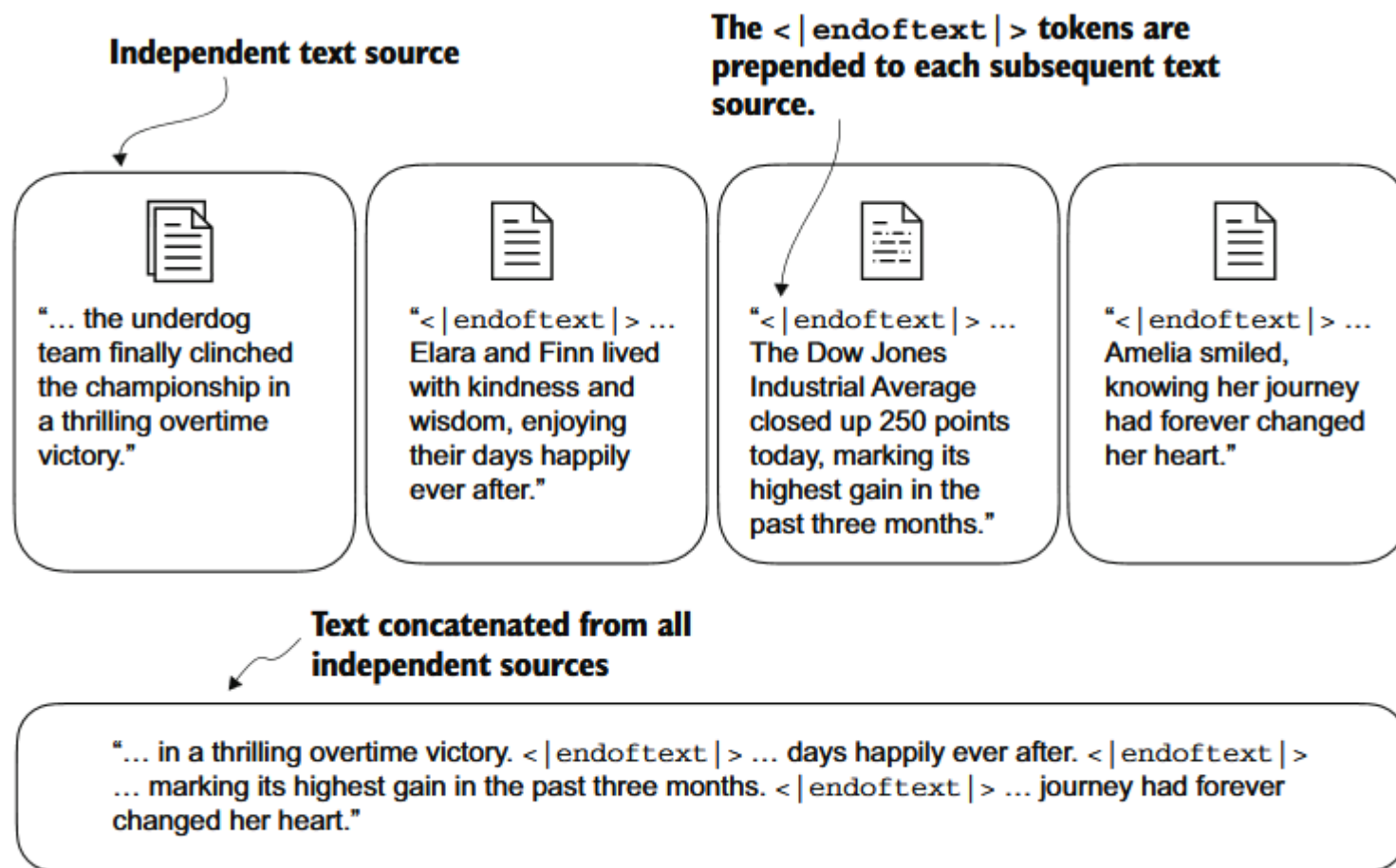


# LLM Tokenization





# LLM Tokenization



# Byte Pair Encoding - BPE

## 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 o k e n s F a l s e N o n e e l i f == >= e l s e : t w o t a b s : " " T h r e e t a b s : " "

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

## 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

## 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

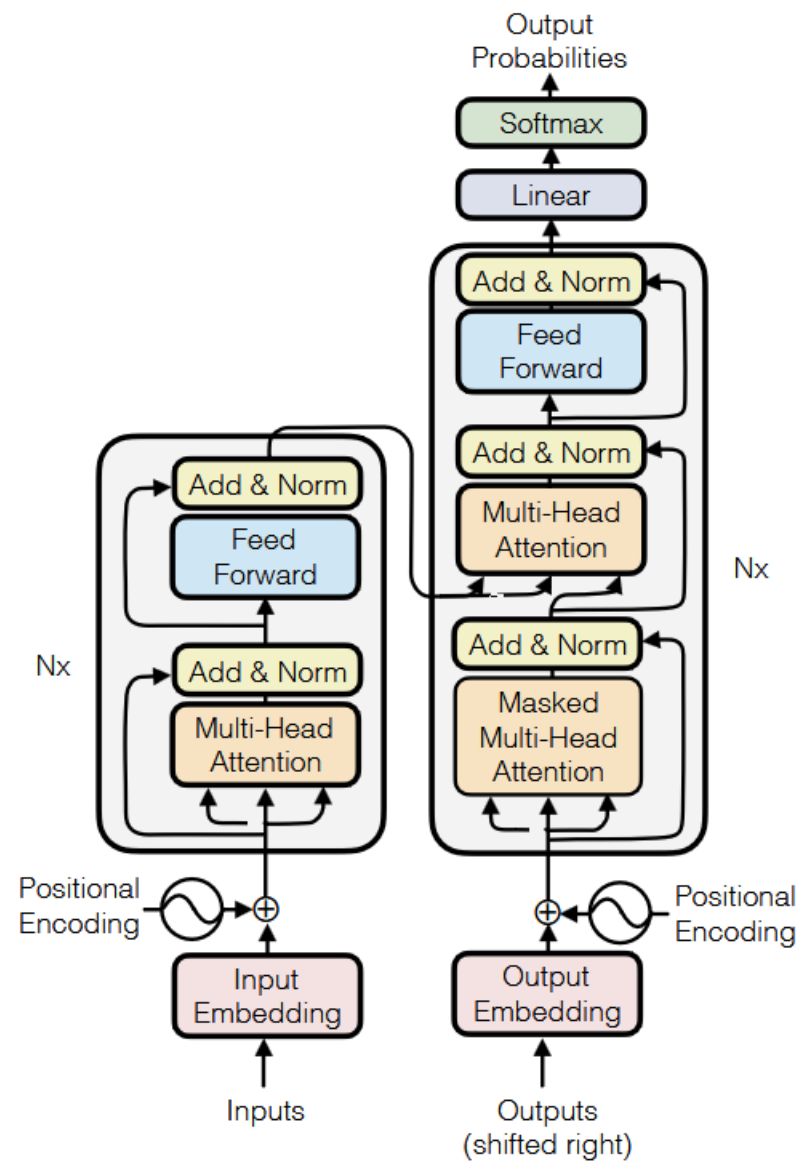
## Trained tokenizer

Tokens	
Token ID	Token
0	!
1	"
...	...
50,257	

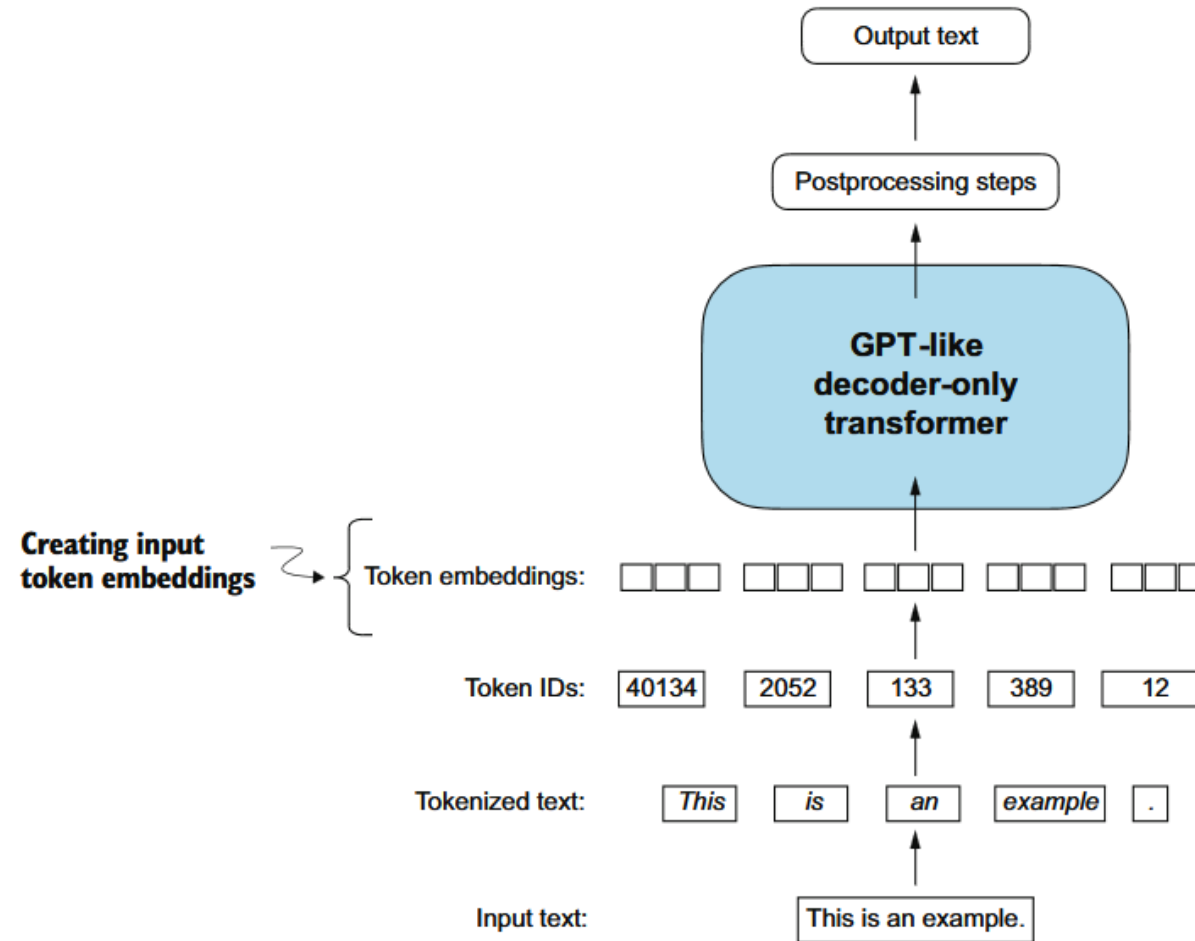
## Language model

Token embeddings	
0	
1	
...	...
50,257	





# Creating token embeddings



# Creating token embeddings

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

```
import torch

torch.manual_seed(123);

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

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

Suppose we want embeddings of size 5

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

# Creating token embeddings

```
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

One-hot encoded ("sparse") representation of "S U N N Y"

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0



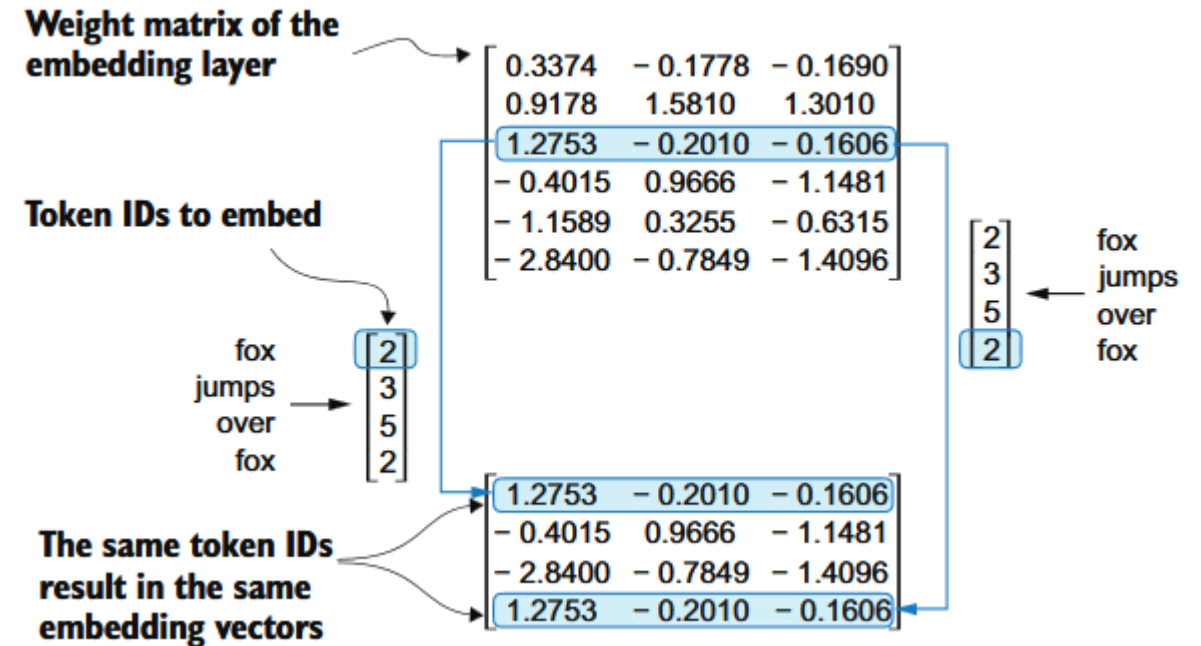
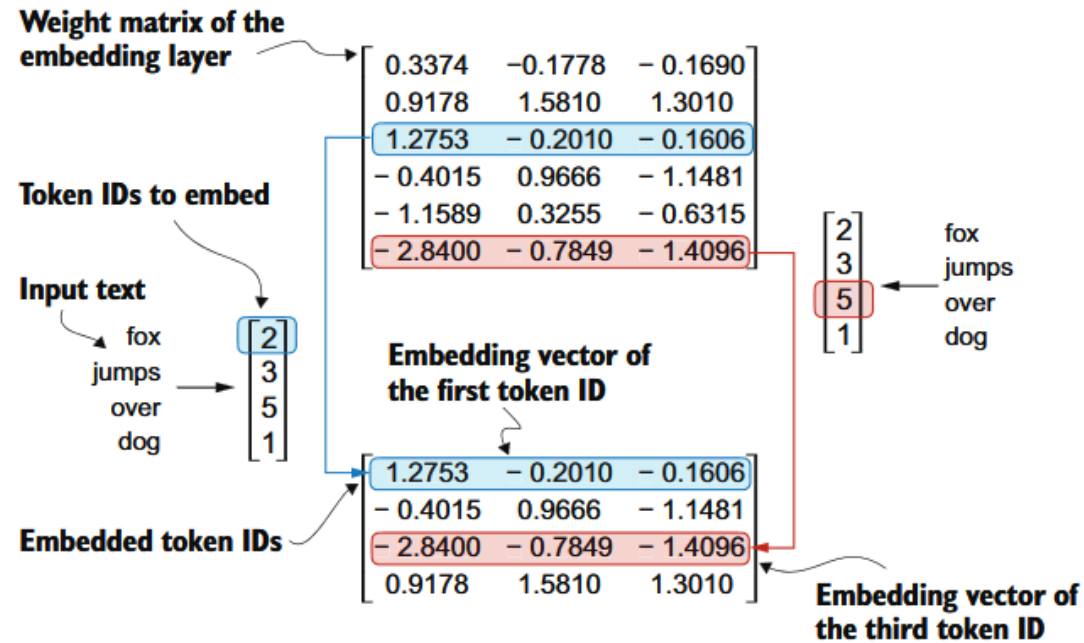
Embedded ("dense")  
representation of  
"SUNNY"

$\begin{bmatrix} 0.9816, & 0.7363, & 0.5899 \\ 0.2605, & 0.3766, & 0.3502 \\ 0.7382, & 0.9807, & 0.4762 \\ 0.6231, & 0.8825, & 0.8836 \end{bmatrix}$

Embedding layer

$\begin{bmatrix} 0.6912, & 0.8765, & 0.4939 \\ 0.6342, & 0.7481, & 0.7717 \\ 0.8395, & 0.2128, & 0.3696 \\ 0.4900, & 0.1509, & 0.0689 \\ 0.2587, & 0.9171, & 0.8670 \\ 0.7213, & 0.9922, & 0.5701 \\ 0.7598, & 0.5231, & 0.3666 \\ 0.5150, & 0.5216, & 0.9682 \\ 0.2248, & 0.0261, & 0.4427 \\ 0.1818, & 0.6863, & 0.8713 \\ 0.4192, & 0.1566, & 0.9004 \\ 0.8102, & 0.5741, & 0.4241 \\ 0.1116, & 0.0466, & 0.2786 \\ \text{S} & 0.9816, & 0.7363, & 0.5899 \\ 0.9224, & 0.3672, & 0.6972 \\ 0.1207, & 0.3372, & 0.2128 \\ 0.0660, & 0.1524, & 0.8440 \\ 0.2162, & 0.5640, & 0.0988 \\ \text{U} & 0.2605, & 0.3766, & 0.3502 \\ 0.7334, & 0.4757, & 0.7581 \\ \text{N} & 0.7382, & 0.9807, & 0.4762 \\ 0.2369, & 0.8102, & 0.8798 \\ 0.6932, & 0.2671, & 0.8018 \\ 0.9593, & 0.5302, & 0.4290 \\ \text{Y} & 0.6231, & 0.8825, & 0.8836 \\ 0.4623, & 0.8503, & 0.7279 \end{bmatrix}$

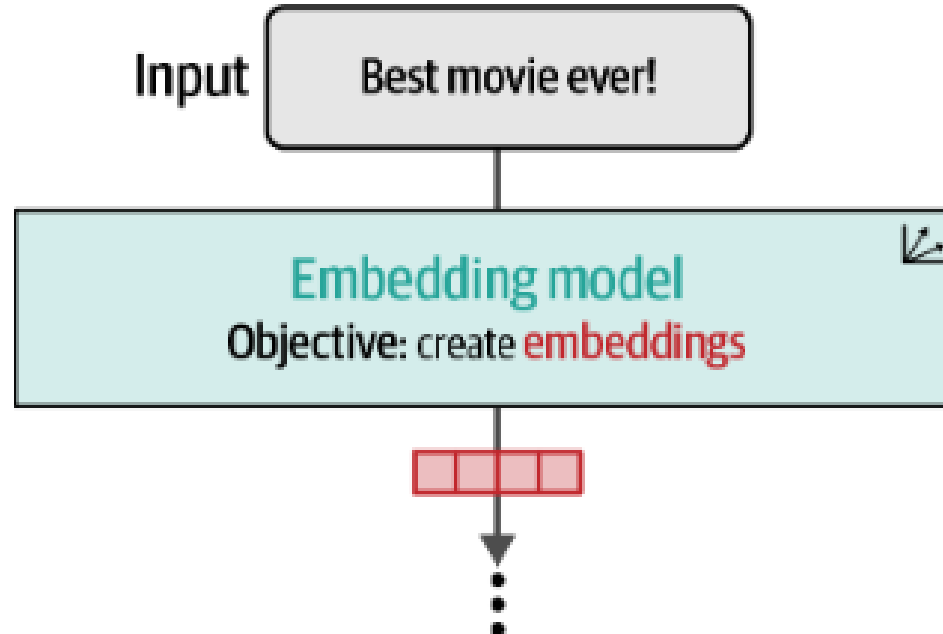
# Creating token embeddings



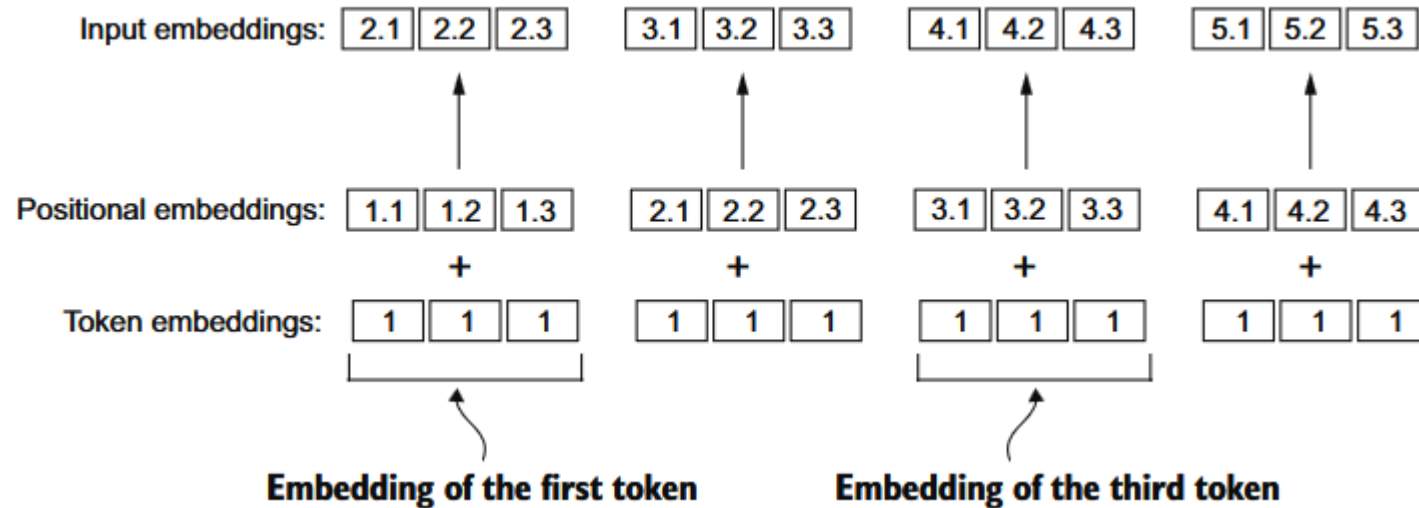


# Sentence embeddings

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

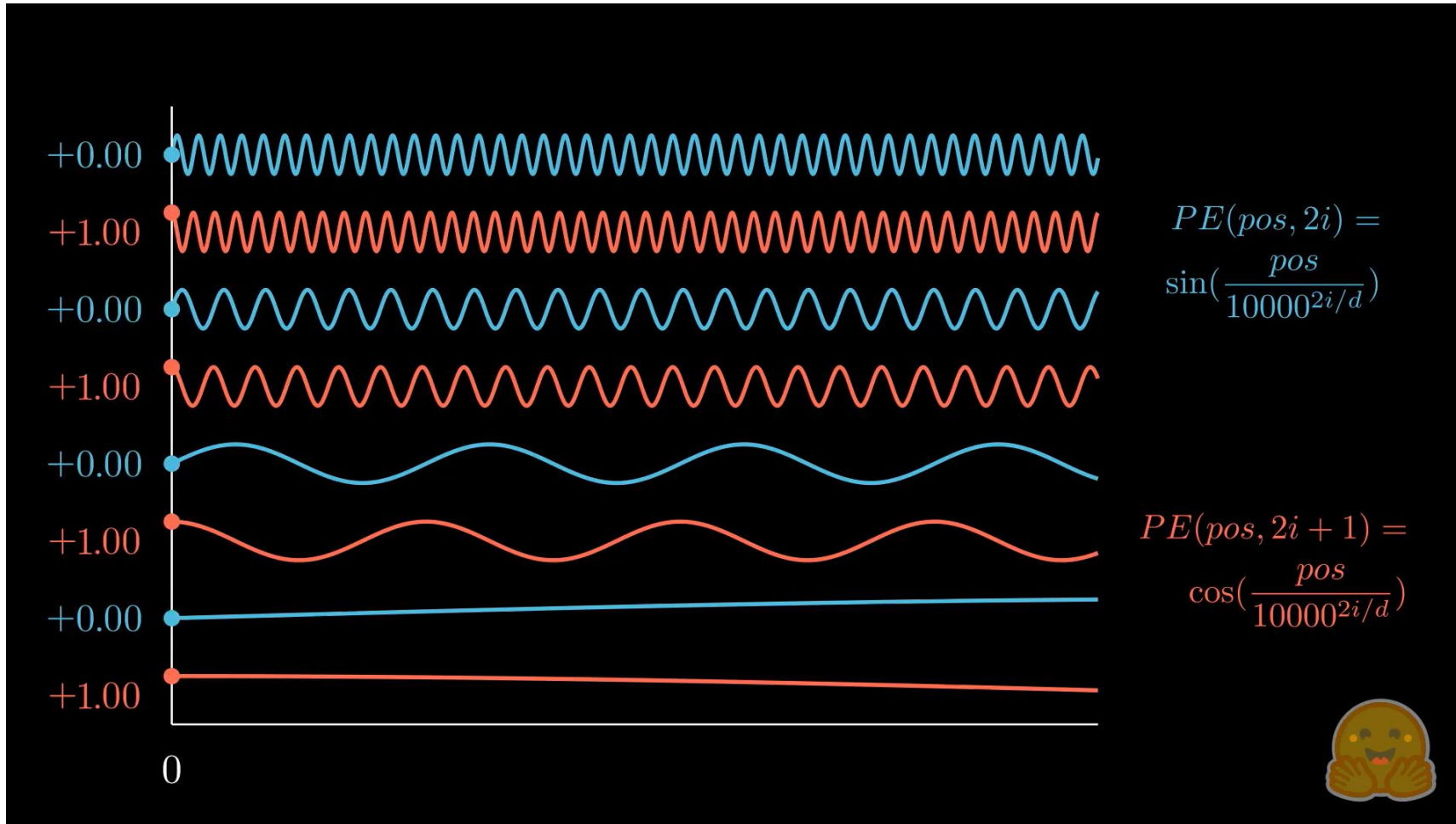


# Encoding word positions



**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)

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

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

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

---

RoFORMER: ENHANCED TRANSFORMER WITH ROTARY  
POSITION EMBEDDING

---

Jianlin Su  
Zhuiyi Technology Co., Ltd.  
Shenzhen  
bojonesu@vezhuiyi.com

Yu Lu  
Zhuiyi Technology Co., Ltd.  
Shenzhen  
julianlu@vezhuiyi.com

Shengfeng Pan  
Zhuiyi Technology Co., Ltd.  
Shenzhen  
nickpan@vezhuiyi.com

Ahmed Murtadha  
Zhuiyi Technology Co., Ltd.  
Shenzhen  
mengjiayi@vezhuiyi.com

Bo Wen  
Zhuiyi Technology Co., Ltd.  
Shenzhen  
brucewen@vezhuiyi.com

Yunfeng Liu  
Zhuiyi Technology Co., Ltd.  
Shenzhen  
glenliu@vezhuiyi.com

$m$  is the token's position in the sequence.

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

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

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

# Encoding word positions

