Working with HF Transformers Library

CS XXX: Introduction to Large Language Models

Contents

- `pipeline` object
- Raw text to prediction
- Tokenizer
- Model
- Postprocessing
- Finetuning
- Loading custom model into `pipeline`

The `pipeline` Object

- The `pipeline` object in Hugging Face is a high-level API designed to simplify working with popular machine learning tasks. It abstracts away much of the complexity involved in loading models, tokenizing inputs, and post-processing outputs. With just a few lines of code, you can use state-of-the-art models for a wide variety of tasks.

```
from transformers import pipeline
```

The `pipeline` Object

- With just a few lines of code, you can use state-of-the-art models for a wide variety of tasks.

  - Sentiment Analysis

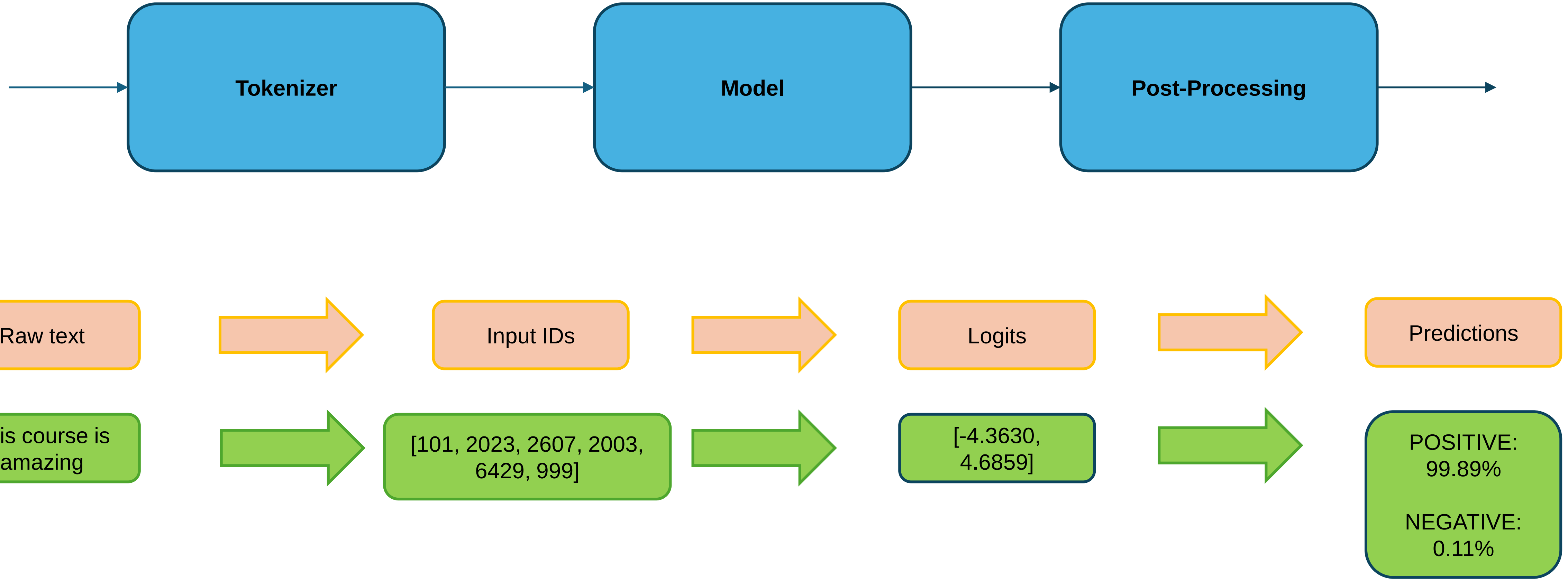
```python
# Import the pipeline function from the transformers library
from transformers import pipeline

# Initialize a pipeline for sentiment analysis
classifier = pipeline("sentiment-analysis")

# Use the classifier to analyze the sentiment of a text
classifier("I love AI and want to know more about the current state of generative AI.")
```

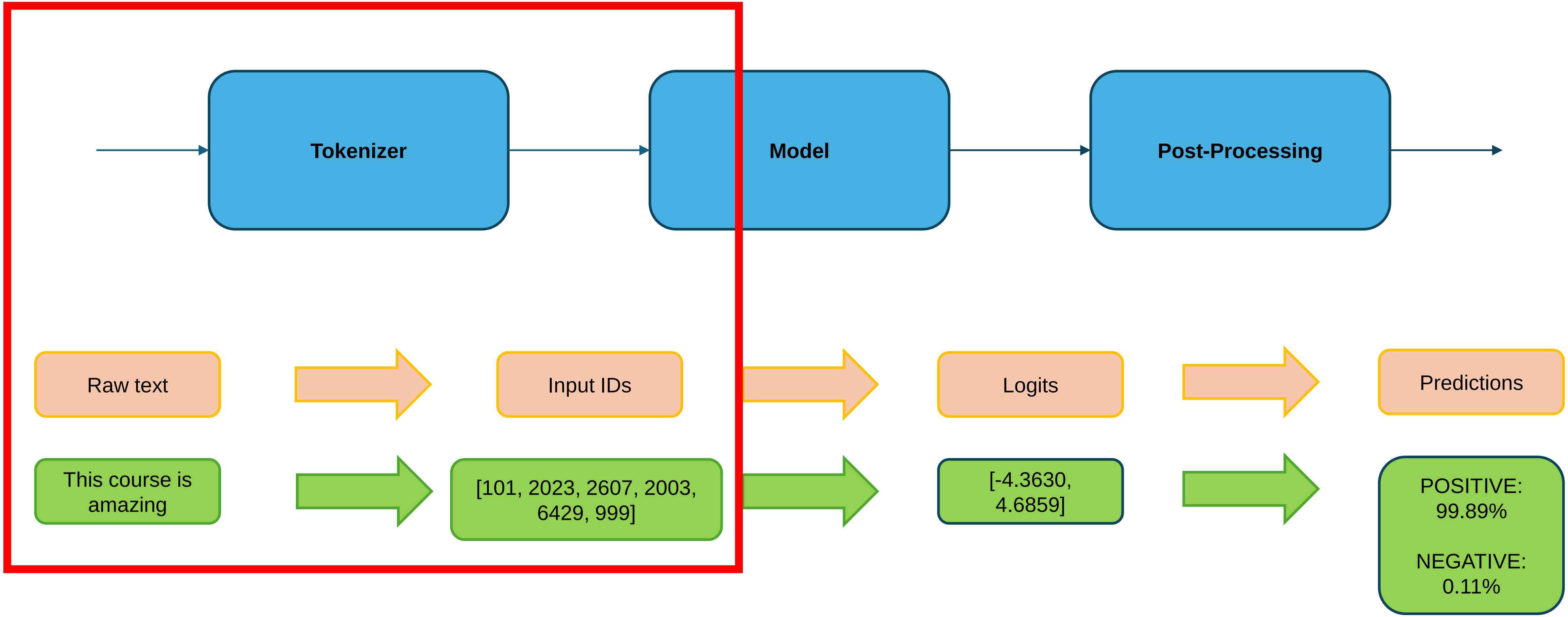
```
No model was supplied, defaulted to distilbert-base-uncased-finetuned-sst-2-English
[{'label': 'POSITIVE', 'score': 0.9966887831687927}]
```

Raw text to Predictions



Tokenizer

- The tokenizer is used to transform raw text to numbers



Tokenizer

- The tokenizer's objective is find a meaningful representation

"The HuggingFace headquarters are based in Brooklyn, New York"

↓

101	3045	5490	234	102

Tokenizer

- Three different ways to tokenize
  - **Word Based:** Splitting a raw text into words
  - **Character Based:** Splitting a raw text into characters
  - **Sub-word Based:** Splitting a raw text into subwords

Tokenizer

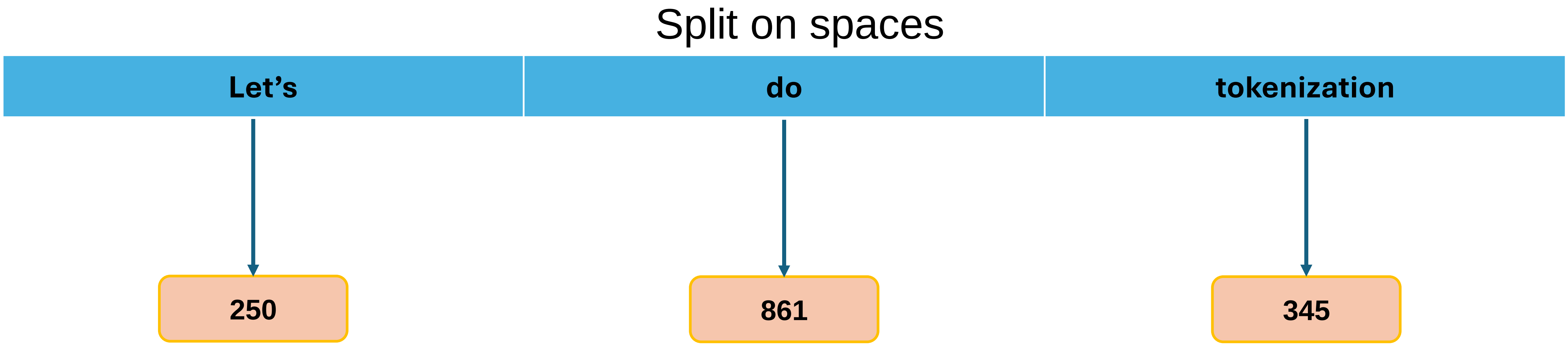
- **Word Based:** Splitting a raw text into words

Split on spaces

Let’s	do	tokenization

Tokenizer

- **Word Based:** Splitting a raw text into words
  - Each word has a specific ID



Tokenizer

- **Word Based:** Splitting a raw text into words
  - Very similar words have entirely different meanings

the	→	1
of	→	2
and	→	3
to	→	4
in	→	5
was	→	6
the	→	7
is	→	8
for	→	9
as	→	10
on	→	11
with	→	12
that	→	13
dog	→	14
dogs	→	15

Tokenizer

- **Word Based:** Splitting a raw text into words
  - The vocabulary can end up being very large

the	→	1
of	→	2
and	→	3
to	→	4
in	→	5
was	→	6
the	→	7
is	→	8
for	→	9
as	→	10
on	→	11
with	→	12
That	→	13
…	→	…
Malapropism	→	**170,000**

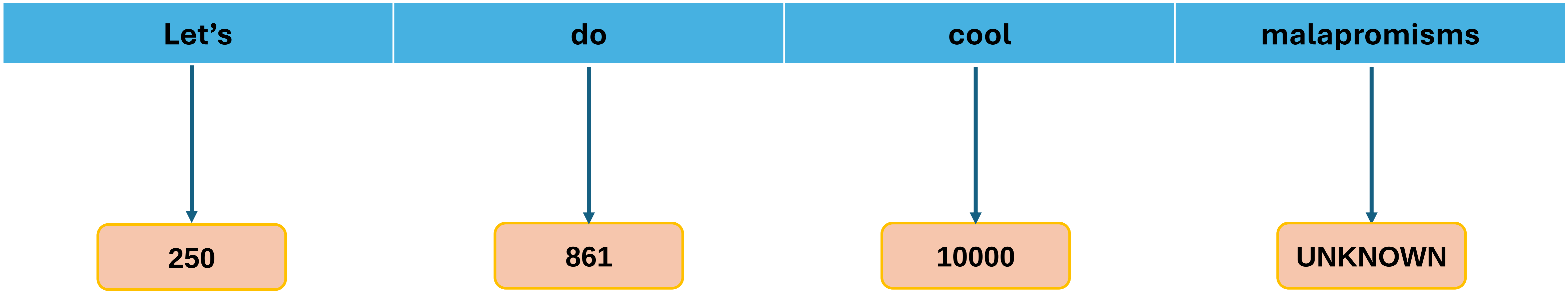
Tokenizer

- **Word Based:** Splitting a raw text into words
  - The vocabulary can end up being very large
  - Large vocabularies result in heavy models
  - We can limit the amount of words we add to the vocabulary

the	→	1
of	→	2
and	→	3
to	→	4
in	→	5
was	→	6
the	→	7
is	→	8
for	→	9
as	→	10
on	→	11
with	→	12
that	→	13
…	→	…
hug	→	**10,000**

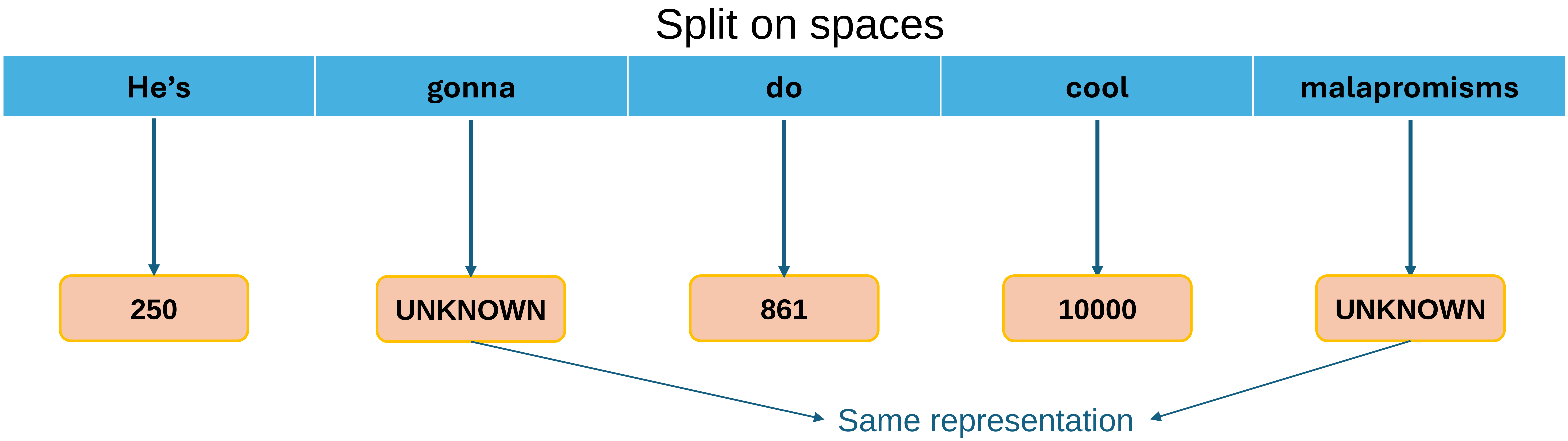
Tokenizer

- **Word Based:** Splitting a raw text into words
  - Out of vocabulary words result in a loss of information



Tokenizer

- **Word Based:** Splitting a raw text into words
  - Out of vocabulary words result in a loss of information



Tokenizer

- **Character Based:** Splitting a raw text into characters
    - Out of vocabulary words result in a loss of information

Split on characters

L	**e**	**t**	**’**	**s**	**d**	**o**	**t**	**o**	**k**	**e**	**n**	**i**	**z**	**t**	**i**	**o**	**n**	**!**

Tokenizer

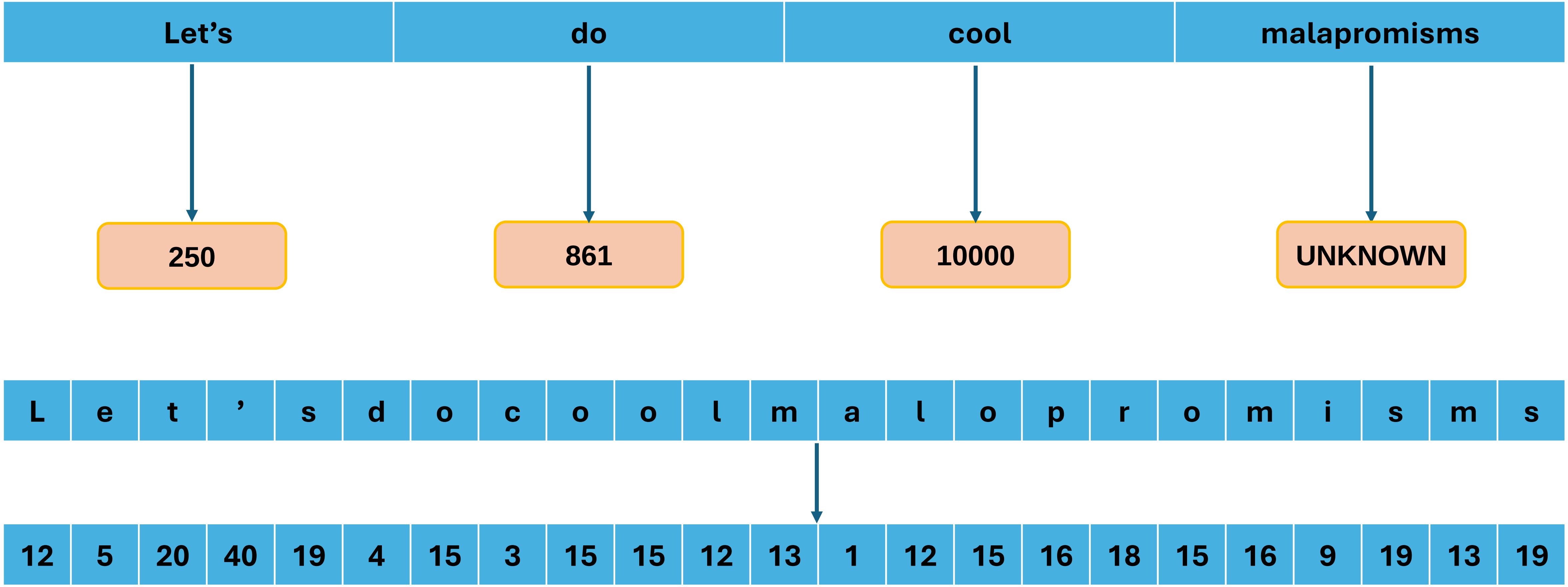
- **Character Based:** Splitting a raw text into characters
  - Vocabularies are slimmer

Character-based vocabulary

a	→	1
b	→	2
c	→	3
d	→	4
e	→	5
f	→	6
g	→	7
…	→	…
1	→	27
2	→	28
3	→	29
…	→	…
!	→	37
…	→	…
à	→	256

Tokenizer

- **Character Based:** Splitting a raw text into characters
  - Fewer out-of-vocabulary words



Tokenizer

- **Sub-word Based:** Splitting a raw text into subwords
  - Middle ground between word and character based algorithms

Word-based vocabulary

Very large vocabularies

Large quantity of out of vocabulary tokens

Loss of meaning across very similar words

Char-based vocabulary

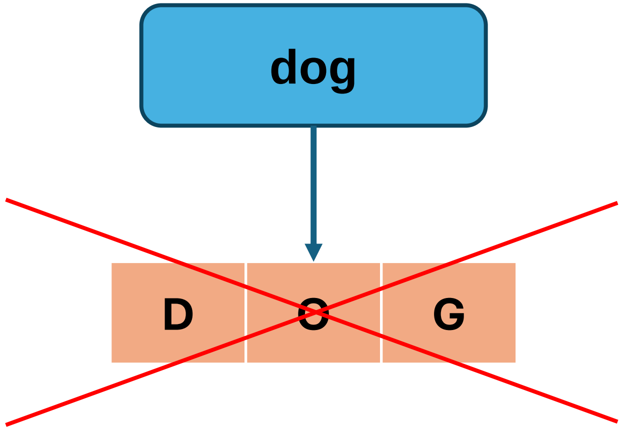
Very long sequences

Less meaningful individual tokens

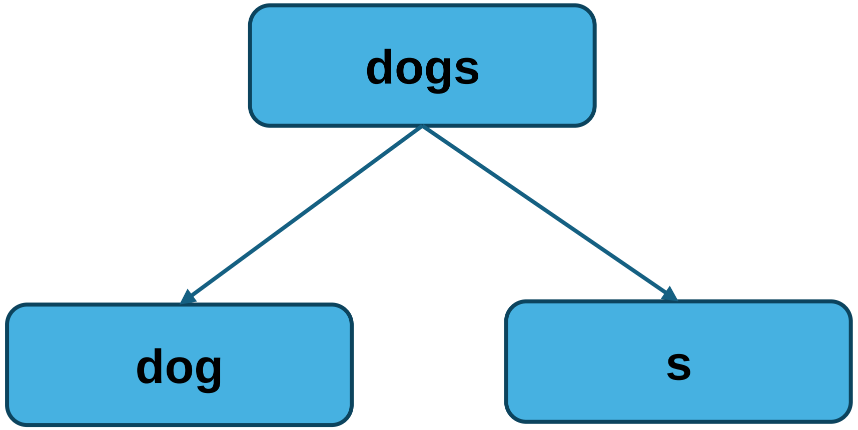
Tokenizer

- **Sub-word Based:** Splitting a raw text into subwords
  - Middle ground between word and character based algorithms

Freqeuently used words should not be split into smaller subwords



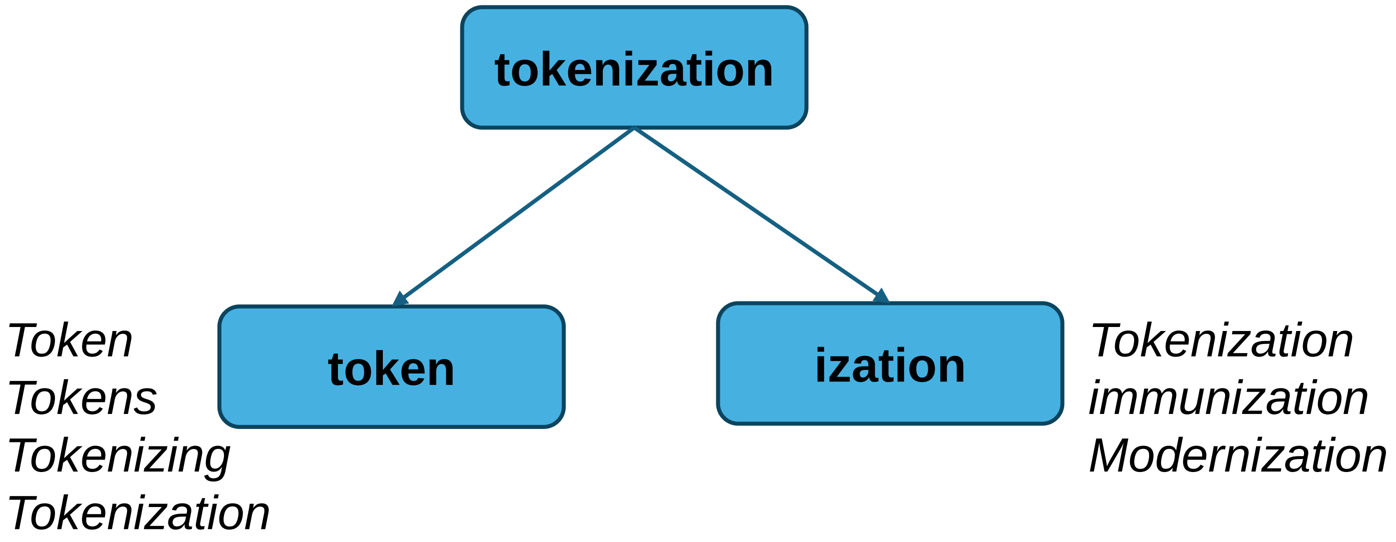
Rare words should be decomposed into meaningful subwords



Tokenizer

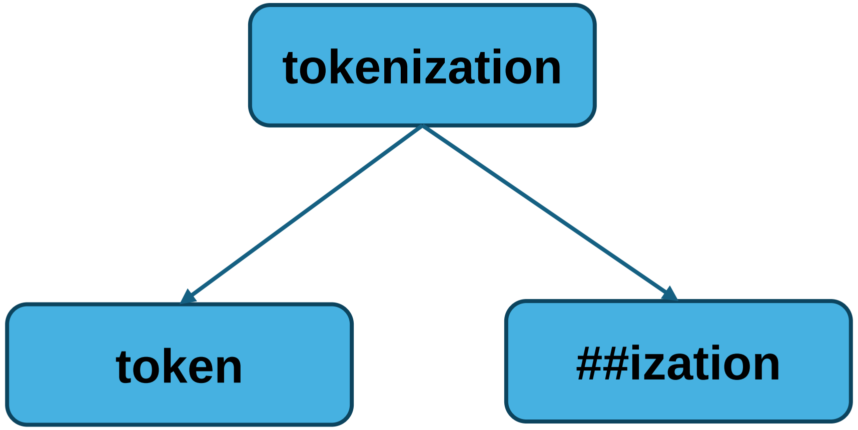
- **Sub-word Based:** Splitting a raw text into subwords
    - Subwords help identify similar syntactic or semantic situations in text

Rare words should be decomposed into meaningful subwords



Tokenizer

- **Sub-word Based:** Splitting a raw text into subwords
  - Subword tokenization algorithms can identify start of word tokens



BERT

  - ## is a marker to indicate that the subword is part of a word and not a standalone token.

Tokenizer

- **Sub-word Based:** Splitting a raw text into subwords
  - Most models obtaining state-of-the-art results in English today use some kind of subword-tokenization algorithm

WordPiece

BERT, DistilBERT

Unigram

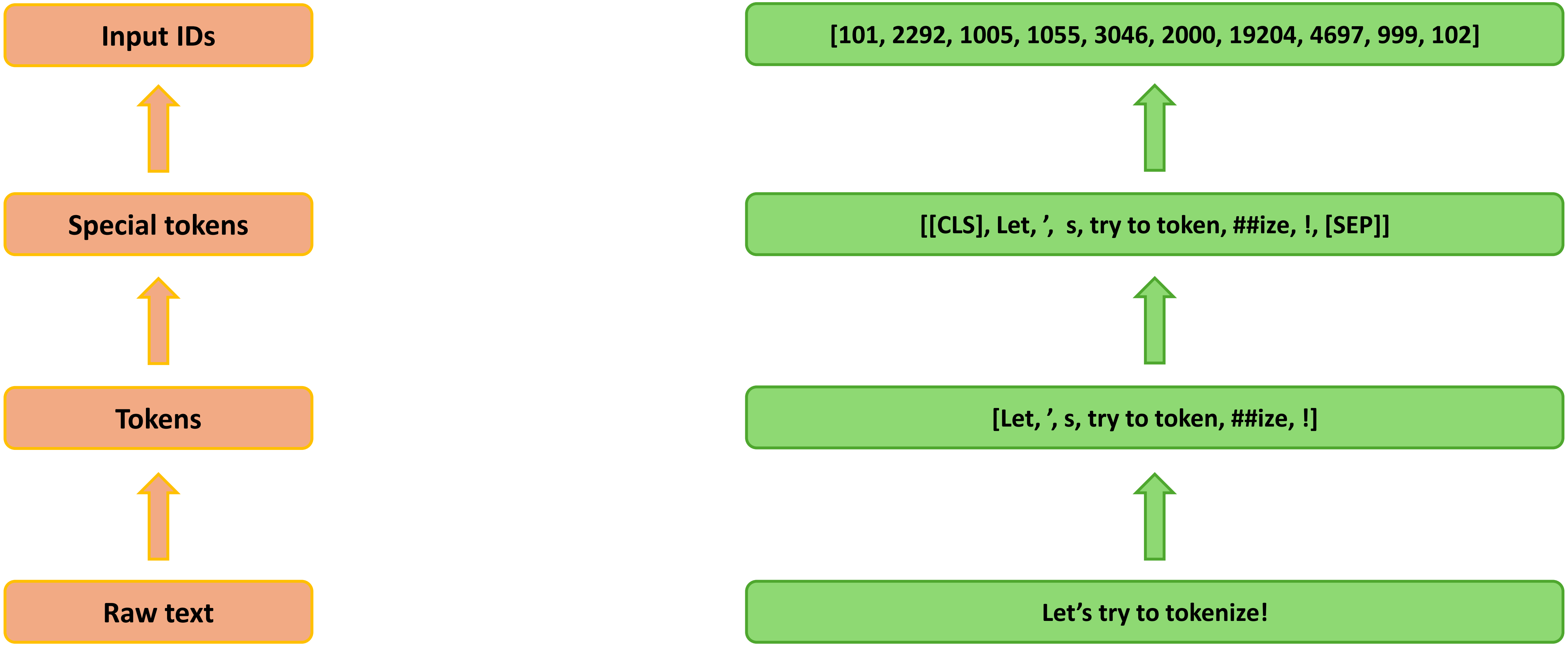
XLNet, ALBERT

Byte-Pair Encoding

GPT2, RoBERTa

Tokenizer

- Tokenization Pipeline



Tokenizer

- The first step of the pipeline is to split the text into tokens

```python
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
tokens = tokenizer.tokenize("Let's try to tokenize!")
print(tokens)
```

```
['let', "'", 's', 'try', 'to', 'token', '##ize', '!']
```

Tokenizer

- Each token has a unique ID

```python
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
tokens = tokenizer.tokenize("Let's try to tokenize!")
input_ids = tokenizer.convert_tokens_to_ids(tokens)
print(input_ids)
```

```
[2292, 1005, 1055, 3046, 2000, 19204, 4697, 999]
```

```python
decoded_string = tokenizer.decode([2292, 1005, 1055, 3046, 2000, 19204, 4697, 999])
print(decoded_string)
```

```
['let', "'", 's', 'try', 'to', 'token', '##ize', '!']
```

Tokenizer

- The tokenizer adds special tokens the model expects

```python
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
tokens = tokenizer.tokenize("Let's try to tokenize!")
input_ids = tokenizer.convert_tokens_to_ids(tokens)
print(input_ids)
final_inputs = tokenizer.prepare_for_model(input_ids)
print(final_inputs["input_ids"])
```

```
[2292, 1005, 1055, 3046, 2000, 19204, 4697, 999]
[101, 2292, 1005, 1055, 3046, 2000, 19204, 4697, 999, 102]
```

```python
print(tokenizer.decode([101, 2292, 1005, 1055, 3046, 2000, 19204, 4697, 999, 102]))
```

```
[‘[CLS]’, 'let', "'", 's', 'try', 'to', 'token', '##ize', '!', ‘[SEP]’]
```

Tokenizer

- A tokenizer takes texts as inputs and outputs numbers the associated model can make sense of

```python
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")

print(tokenizer("Using a Transformer network is simple"))
```

```
{'input_ids': [101, 7993, 170, 11303, 1200, 2443, 1110, 3014, 102],
 'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0],
 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

Tokenizer

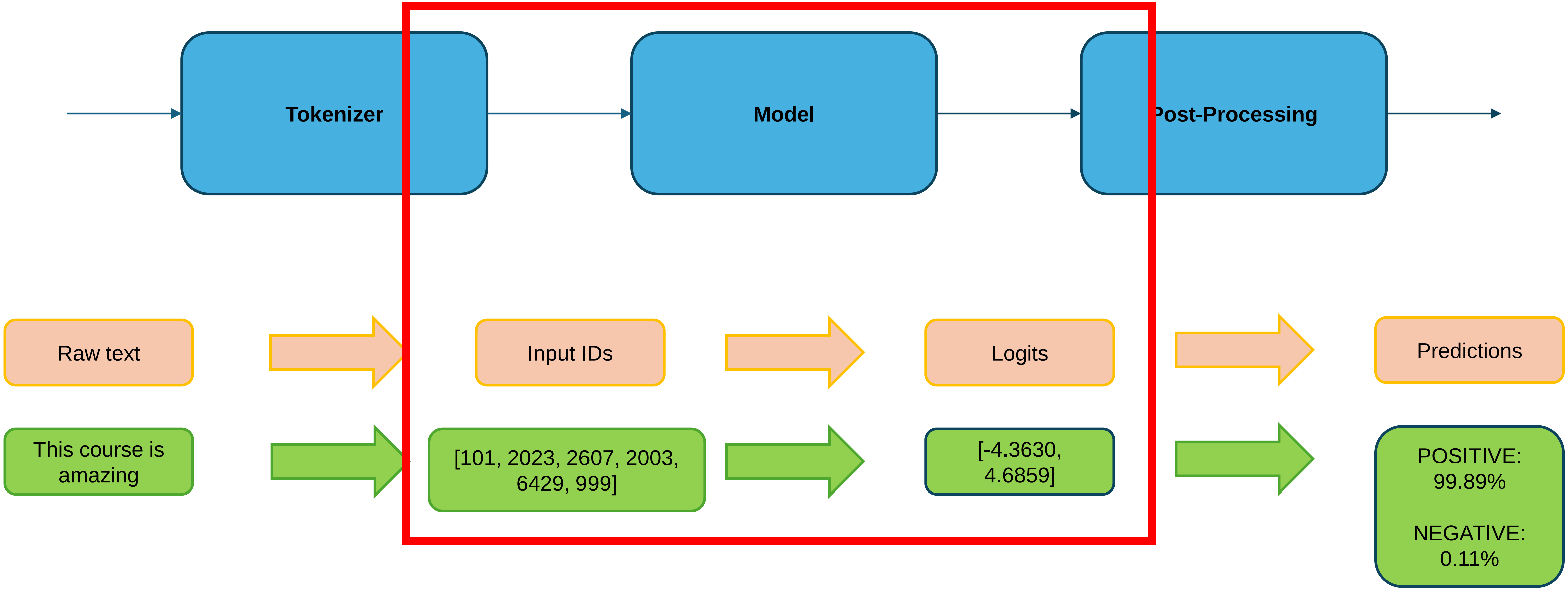
- A tokenizer takes texts as inputs and outputs numbers the associated model can make sense of

```python
from transformers import AutoTokenizer
checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
raw_inputs = [
    "I've been waiting for a HuggingFace course my whole life.",
    "I hate this so much!",
]
inputs = tokenizer(raw_inputs, padding=True, truncation=True, return_tensors="pt")
print(inputs)
```

```
{
'input_ids': tensor([
[ 101, 1045, 1005, 2310, 2042, 3403, 2005, 1037, 17662, 12172, 2607, 2026, 2878, 2166, 1012, 102],
[ 101, 1045, 5223, 2023, 2061, 2172, 999, 102, 0, 0, 0, 0, 0, 0, 0, 0]
]),
'attention_mask': tensor([
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
])
}
```

Model

- The model takes the input IDs and processes them through its layers to generate logits. Logits are unnormalized scores representing the likelihood of different outcomes.



Model

- The AutoModel class loads a model without a task-specific pretrained head

```python
from transformers import AutoModel
checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
model = AutoModel.from_pretrained(checkpoint)
outputs = model(**inputs)
print(outputs.last_hidden_state.shape)
```

```
torch.Size([2, 16, 768])
```

batch size

sequence length

hidden size

Model

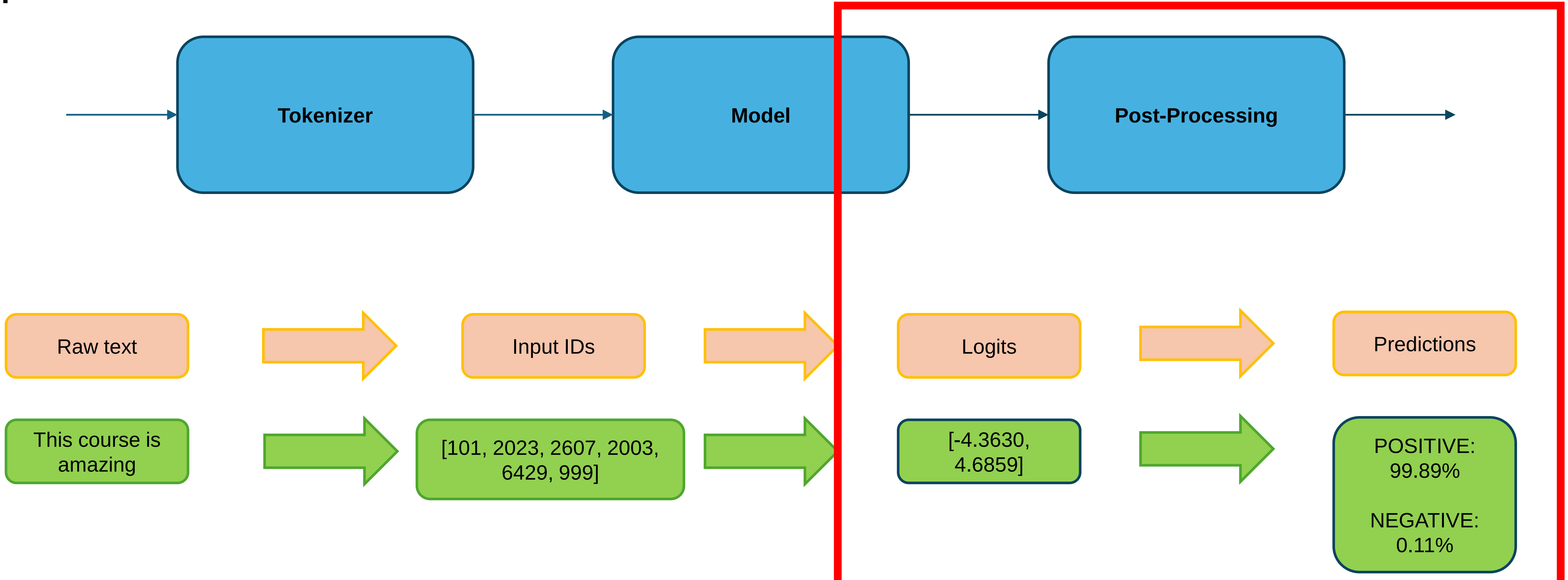
- Each AutoModelFor X class loads a model suitable for a specific task

```python
from transformers import AutoModelForSequenceClassification
checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
model = AutoModelForSequenceClassification.from_pretrained(checkpoint)
outputs = model(**inputs)
print(outputs.logits)
```

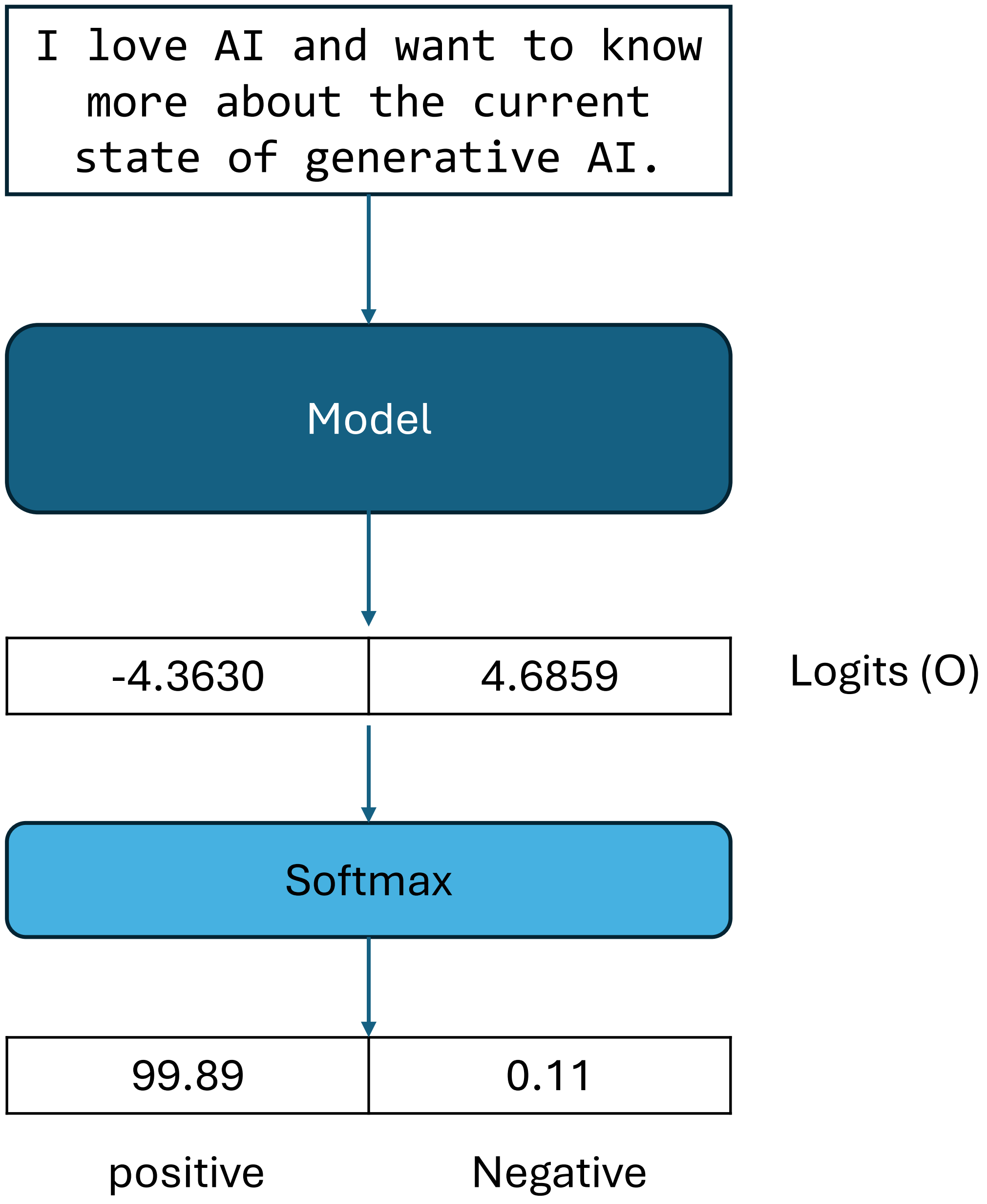
```
tensor([[-1.5607,  1.6123],
        [ 4.1692, -3.3464]], grad_fn=<AddmmBackward>)
```

Post-Processing

- The logits are transformed into meaningful predictions. This involves normalizing the logits using methods like softmax to produce probabilities



Post-Processing



Finetuning

1. Load model for finetuning

```python
from transformers import AutoModelForSequenceClassification, AutoTokenizer
# 1. Load the model and tokenizer
model_name = "bert-base-uncased"
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)  #
Binary classification
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

Finetuning

2. Prepare Data

```python
from datasets import load_dataset
# 2. Prepare the dataset
# Example dataset: SST2 for sentiment classification
dataset = load_dataset("glue", "sst2")

# Tokenize the dataset
def tokenize_function(examples):
    return tokenizer(examples['sentence'], padding="max_length", truncation=True)

tokenized_datasets = dataset.map(tokenize_function, batched=True)

# Split into train, validation, and test datasets
train_dataset = tokenized_datasets["train"]
val_dataset = tokenized_datasets["validation"]
test_dataset = tokenized_datasets["test"]
```

Finetuning

3. Define Training Arguments

```
from transformers import TrainingArguments
# 3. Define Training Arguments
training_args = TrainingArguments(
    output_dir="./results",          # output dir for model predictions & checkpoints
    evaluation_strategy="epoch",     # Evaluate every epoch
    learning_rate=2e-5,              # Learning rate
    per_device_train_batch_size=16,  # Batch size per GPU/CPU for training
    per_device_eval_batch_size=64,   # Batch size for evaluation
    num_train_epochs=3,              # Number of training epochs
    weight_decay=0.01,               # Strength of weight decay
    logging_dir="./logs",            # Directory for storing logs
    logging_steps=10,                # Log every 10 steps
    load_best_model_at_end=True,     # Load the best model when finished training
    metric_for_best_model="accuracy",# Metric to use for best model
)
```

Finetuning

4. Train the model and save

```python
from transformers import Trainer
# Define Trainer
trainer = Trainer(
    model=model,                        # The model to train
    args=training_args,                 # Training arguments
    train_dataset=train_dataset,        # Training dataset
    eval_dataset=val_dataset,           # Evaluation dataset
    tokenizer=tokenizer,                # Tokenizer
)

# 4. Train the model
trainer.train()

# Save the fine-tuned model
trainer.save_model("./fine_tuned_bert")
tokenizer.save_pretrained("./fine_tuned_bert")
```

Finetuning

5. Test the model

```
# 5. Testing the model on the test set
test_results = trainer.evaluate(eval_dataset=test_dataset)
print("Test Results:", test_results)
```

Loading custom model or tokenizer into `pipeline`

- If you have a custom model or tokenizer, you can load it into `pipeline` for inference as follows

```
from transformers import pipeline, AutoModelForSequenceClassification, AutoTokenizer

# Load a custom model and tokenizer
model_path = "./fine_tuned_bert"  # This is where your model and tokenizer are saved
model = AutoModelForSequenceClassification.from_pretrained(model_path)
tokenizer = AutoTokenizer.from_pretrained(model_path)

# Initialize the pipeline with the custom model and tokenizer
classifier = pipeline("text-classification", model=model, tokenizer=tokenizer)

# Use the pipeline
text = "This is a fantastic example!"
output = classifier(text)
print(output)
```

References

- Hugging Face. The Hugging Face Natural Language Processing Course.