CS 335: Introduction to Large Language Models

HuggingFace

Week 6

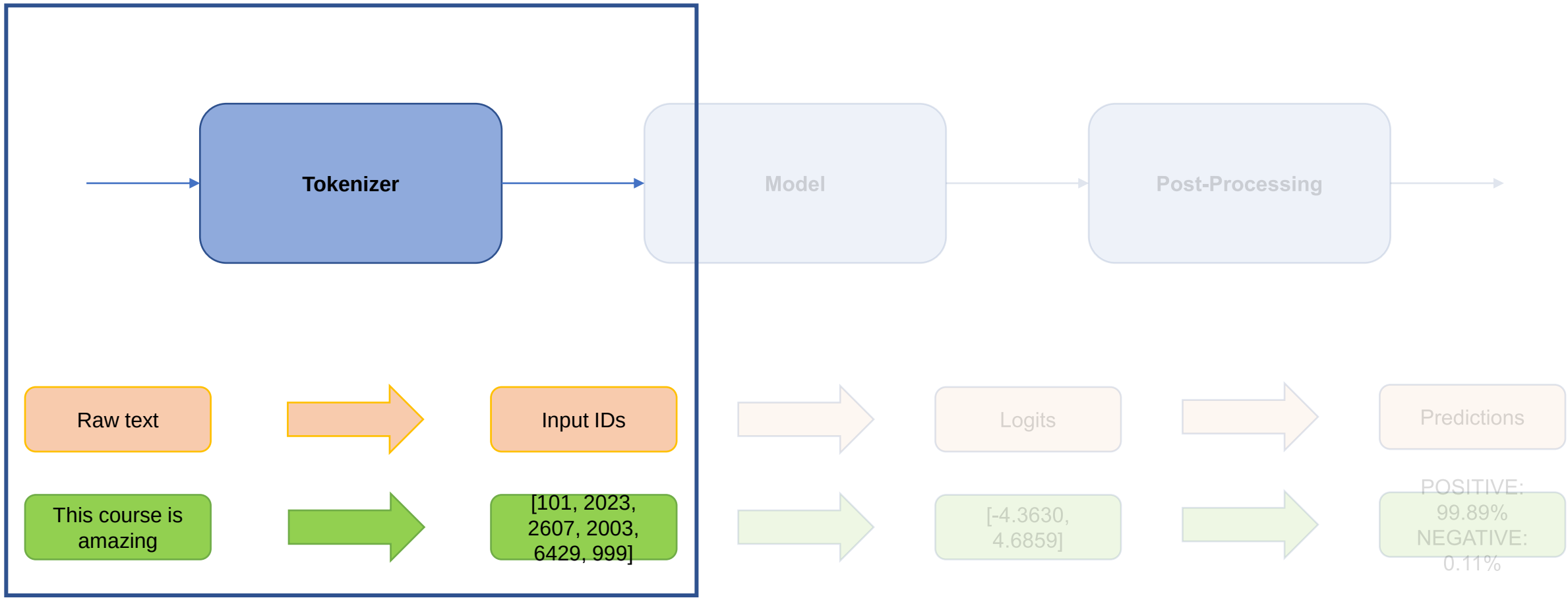
Dr Abdul Samad
Dr Faisal Alvi

Lecture Outline

- Tokenizer
- Word based tokenization
- Char based tokenization
- Sub-word based tokenization
- Model
- Post-Processing
- Softmax

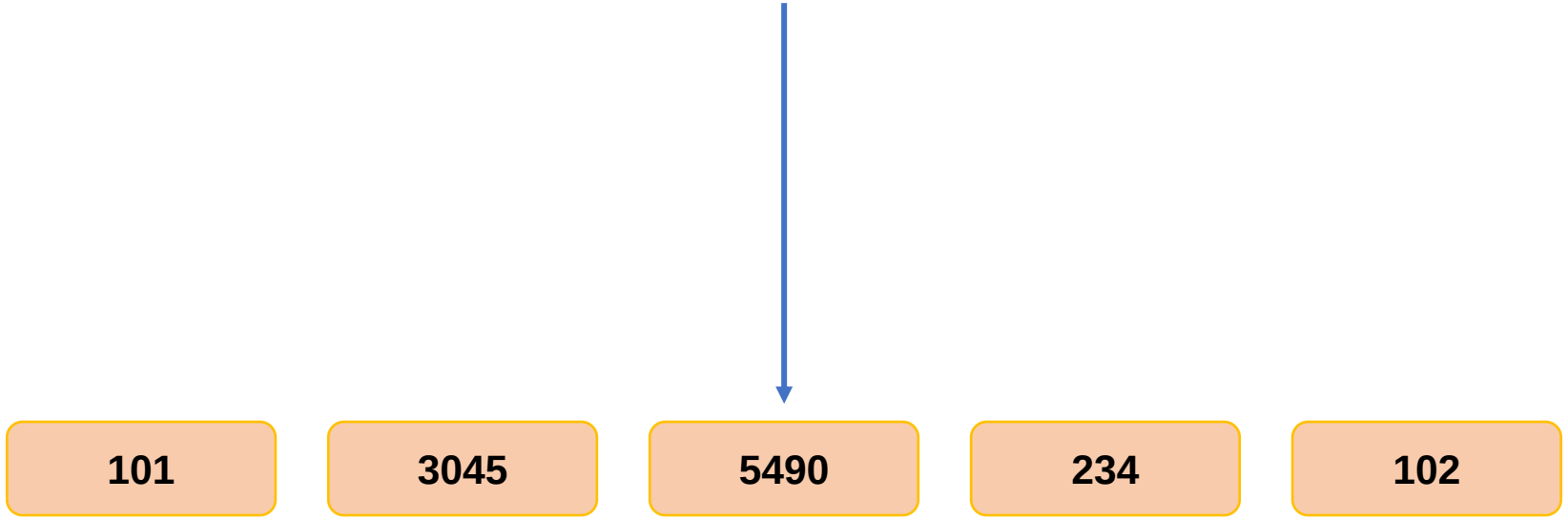
“The course is amazing”

In NLP, most of the data is raw text



The tokenizer is used to transform raw text to numbers

“The HuggingFace headquarters are based in Brooklyn, New York”



The tokenizer’s objective is to find a meaningful representation

Word based

Character based

Subword based

We’ll take a look at three different algorithms

Word-based tokenization

Splitting a raw text into words

Split on spaces

Let’s	do	tokenization

The text is split on spaces

Split on spaces

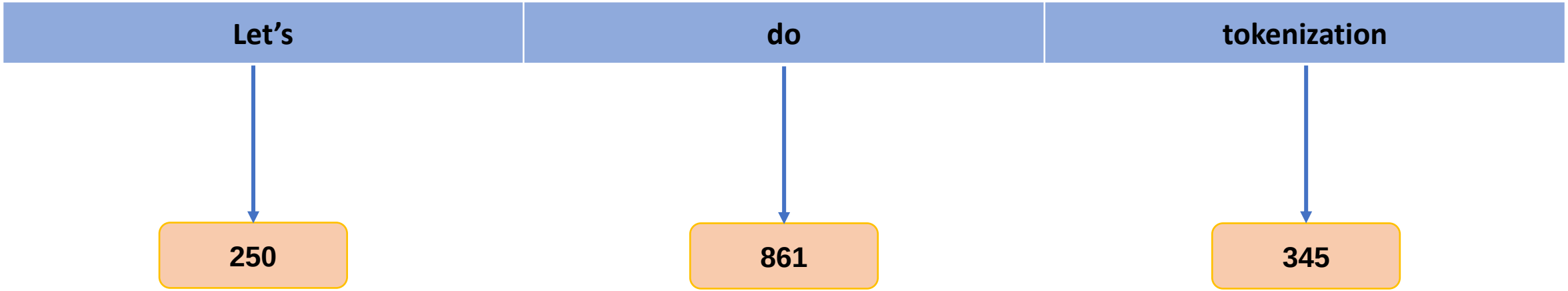
Let’s	do	tokenization

Split on punctuation

Let	’s	do	tokenization	!

Other rules, such as punctuation, may be added

Split on spaces



Each word has a specific ID

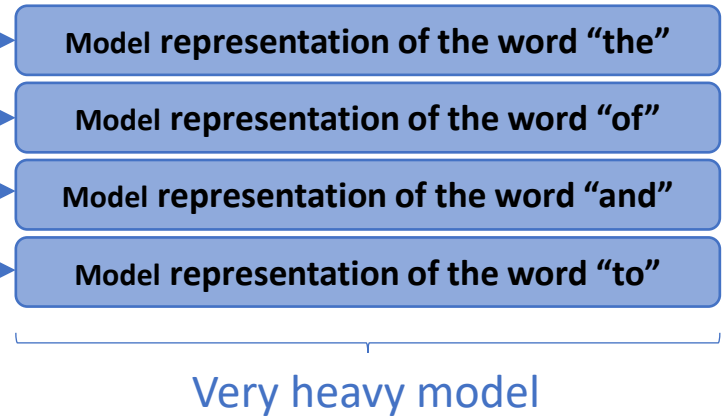
the	→	1
of	→	2
and	→	3
to	→	4
in	→	5
was	→	6
the	→	7
is	→	8
for	→	9
as	→	10
on	→	11
with	→	12
that	→	13
dog	→	14
dogs	→	15

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
…	→	…
Malapropism	→	170,000

The vocabulary can end up very large

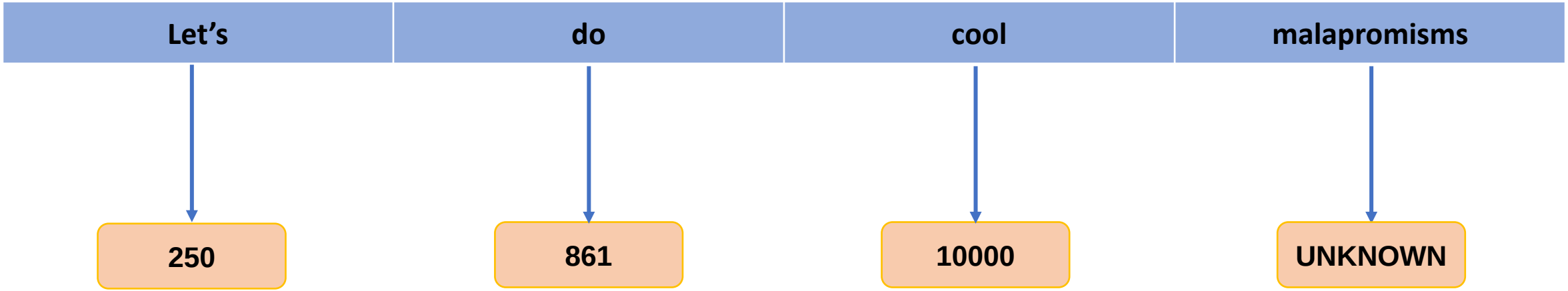
the	→	1
of	→	2
and	→	3
to	→	4
in	→	5
was	→	6
the	→	7
is	→	8
for	→	9
as	→	10
on	→	11
with	→	12
That	→	13
…	→	…
Malapropism	→	170,000



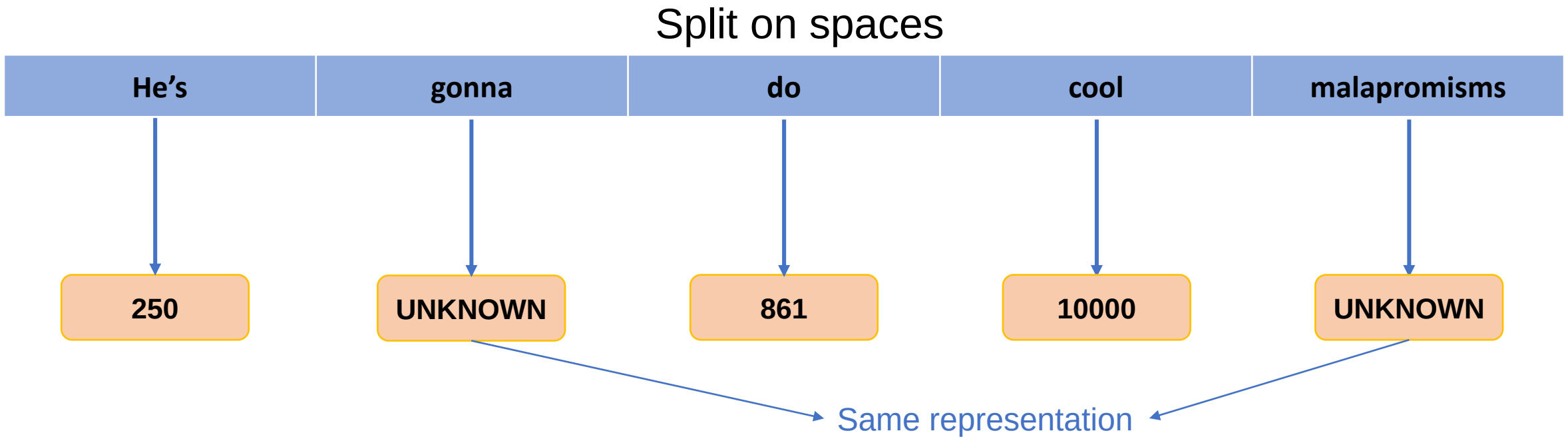
Large vocabularies result in heavy models

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

We can limit the amount of words we add to the vocabulary



Out of vocabulary words result in a loss of information



Out of vocabulary words result in a loss of information

Character-based tokenization

Splitting a raw text into characters

Split on characters

The text is split on characters

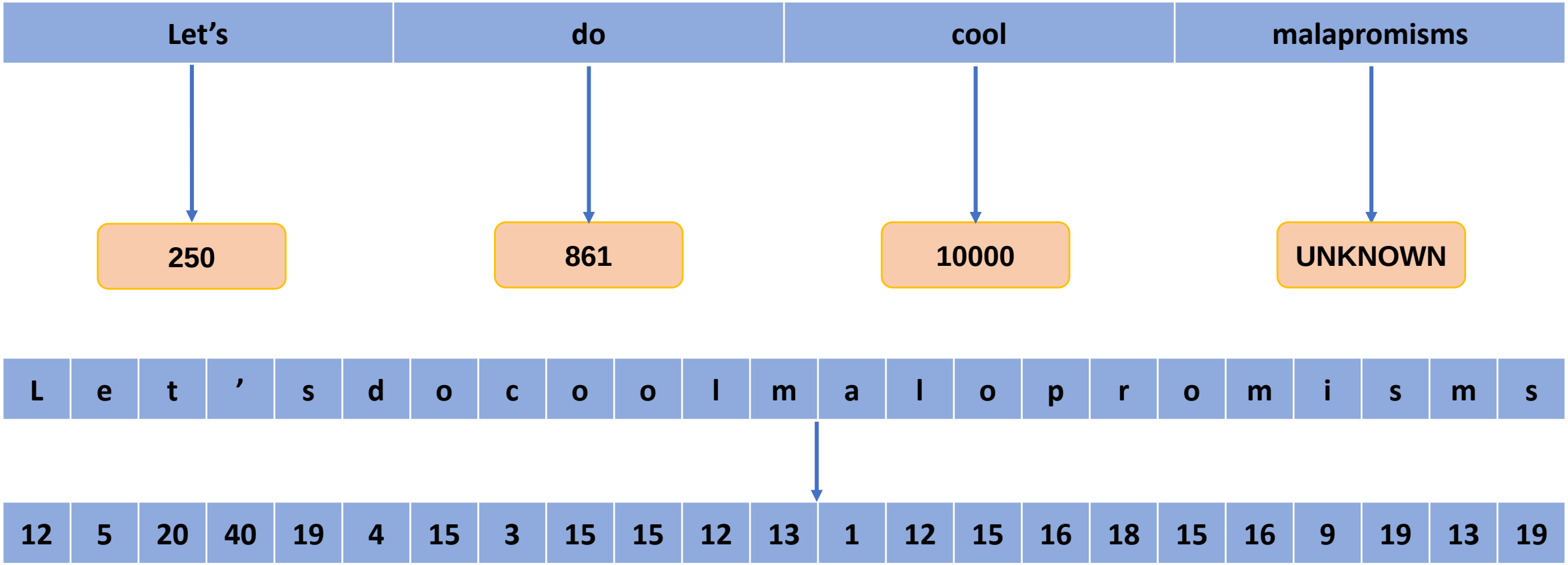
a	→	1
b	→	2
c	→	3
d	→	4
e	→	5
f	→	6
g	→	7
…	→	…
1	→	27
2	→	28
3	→	29
…	→	…
!	→	37
…	→	…
à	→	256

Character-based 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
…	→	…
Malapropism	→	170,000

Word-based vocabulary

Vocabularies are slimmer



Fewer out of vocabulary words

Word-based vocabulary

250	861	10000	UNKNOWN

Character-based vocabulary

12	5	20	40	19	4	15	3	15	15	12	13	1	12	15	16	18	15	16	9	19	13	19

Fewer out of vocabulary words

Subword-based tokenization

Splitting a raw text into subwords

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

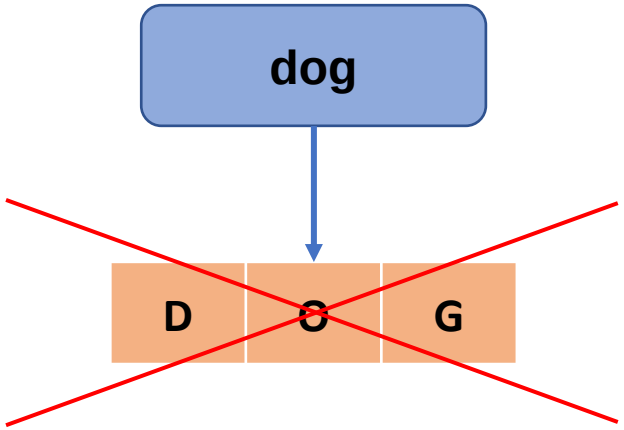
Finding a middle ground between word and character-based algorithms

- ***Freqeuently used words should not be split into smaller subwords***

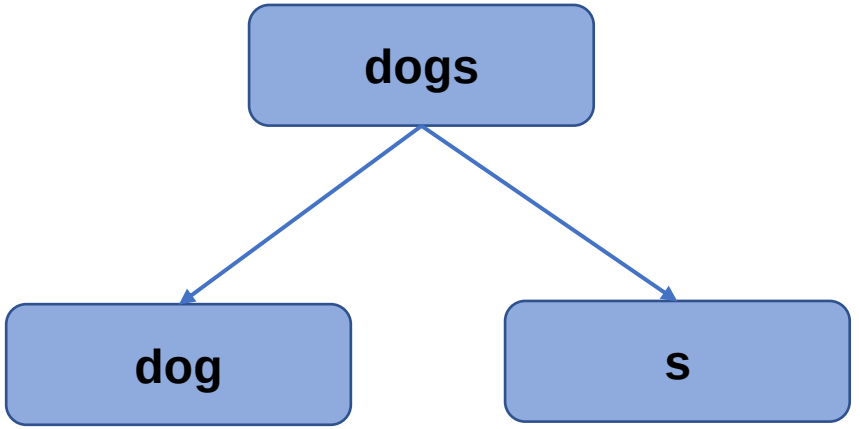
- ***Rare words should be decomposed into meaningful subwords***

Subword-based tokenization lies between character and word-based algorithms

Freqeuently used words should not be split into smaller subwords

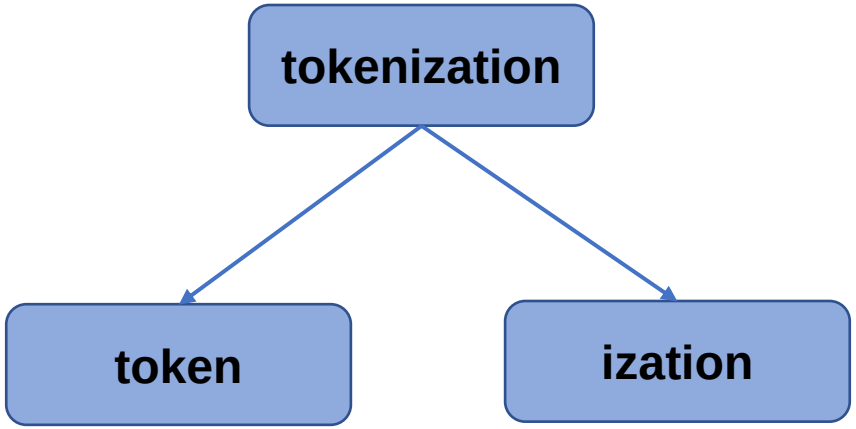


Rare words should be decomposed into meaningful subwords



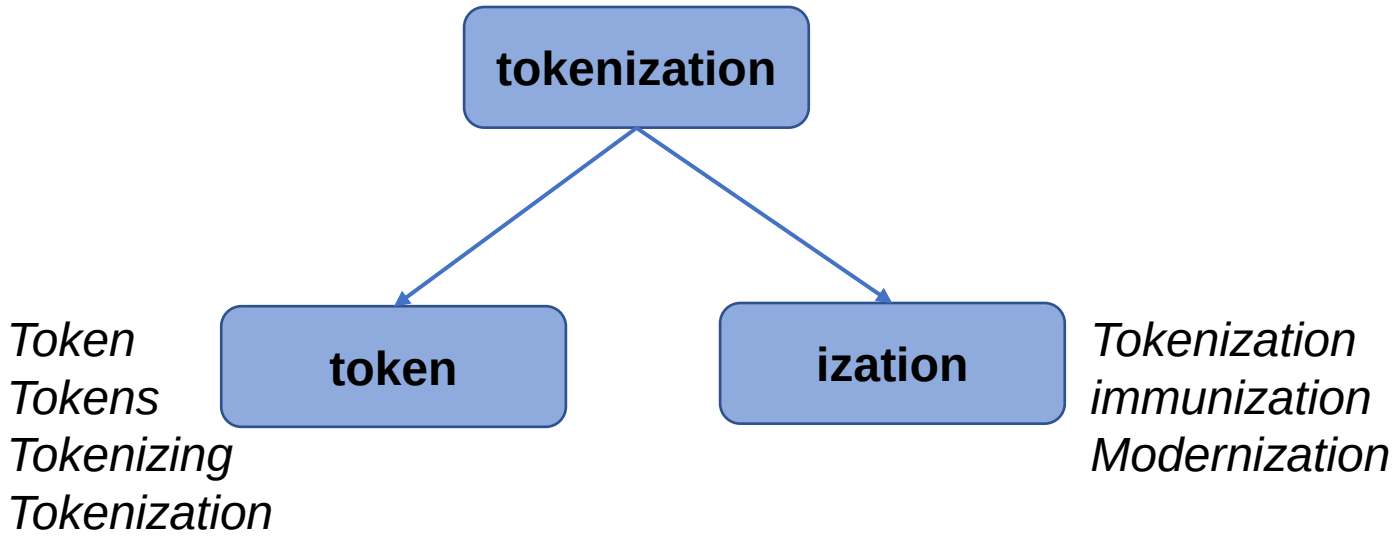
Subword-based tokenization lies between character and word-based algorithms

Rare words should be decomposed into meaningful subwords

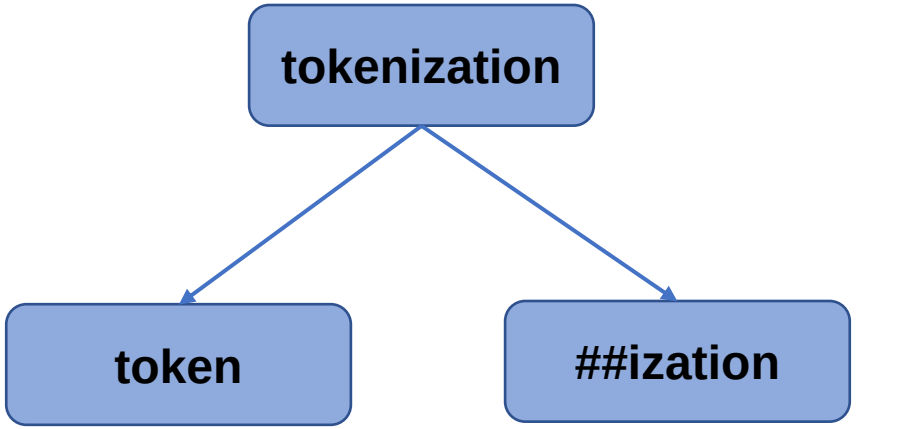


Rare words should be decomposed into meaningful subwords

Rare words should be decomposed into meaningful subwords



Subwords help identify similar syntactic or semantic situations in text



BERT

Subword tokenization algorithms can identify start of word tokens

WordPiece

BERT, DistilBERT

Unigram

XLNet, ALBERT

Byte-Pair Encoding

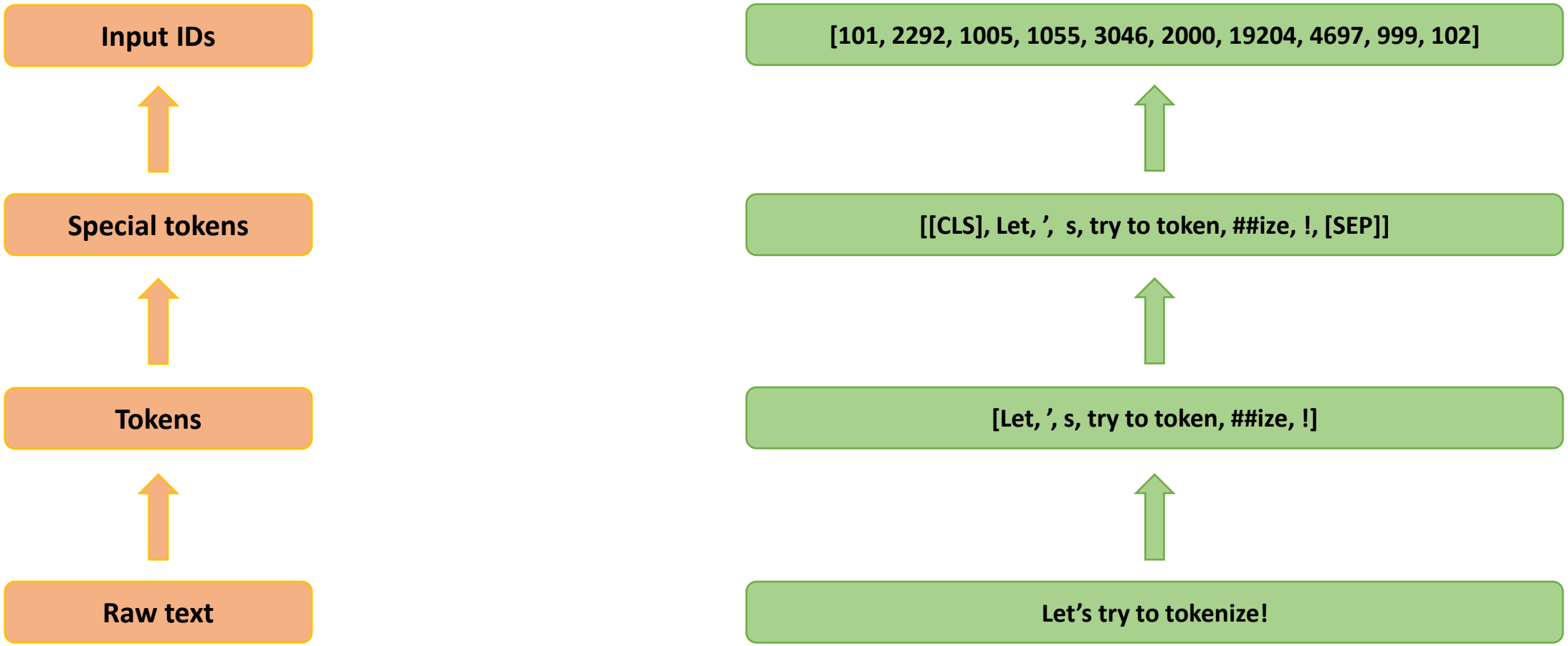
GPT2, RoBERTa

Most models obtaining state-of-the-art results in English today use some kind of subword-tokenization algorithm

```python
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
inputs = tokenizer("Let's try to tokenize!")
print(inputs["input_ids"])
```

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

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



The tokenization pipeline: from input text to a list of numbers

```
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', '!']
```

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

```
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', '!']
```

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("albert-base-v1")
tokens = tokenizer.tokenize("Let's try to tokenize!")
print(tokens)
```

```
['▁let', "'", 's', '▁try', '▁to', '▁to', 'ken', 'ize', '!']
```

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

```
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]
```

Lastly, the tokenizer adds special tokens the model expects

```
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]
```

Lastly, 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)
```

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

```python
final_inputs = tokenizer.prepare_for_model(input_ids)
print(final_inputs["input_ids"])
```

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

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

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
inputs = tokenizer("Let's try to tokenize!")

print(tokenizer.decode(inputs["input_ids"]))
```

```
[CLS] let's try to tokenize! [SEP]
```

The decode method allows us to check how the final output of the tokenizer translates back into text

```python
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
inputs = tokenizer("Let's try to tokenize!")

print(tokenizer.decode(inputs["input_ids"]))
```

```
[CLS] let's try to tokenize! [SEP]
```

```python
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("roberta-base")
inputs = tokenizer("Let's try to tokenize!")

print(tokenizer.decode(inputs["input_ids"]))
```

```
<s>Let's try to tokenize!</s>
```

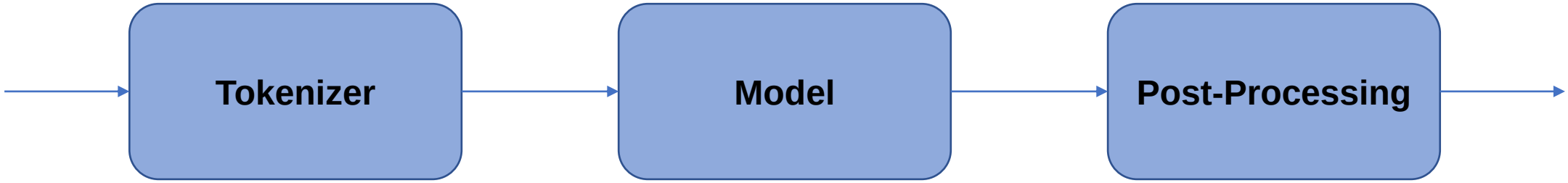
The decode method allows us to check how the final output of the tokenizer translates back into text

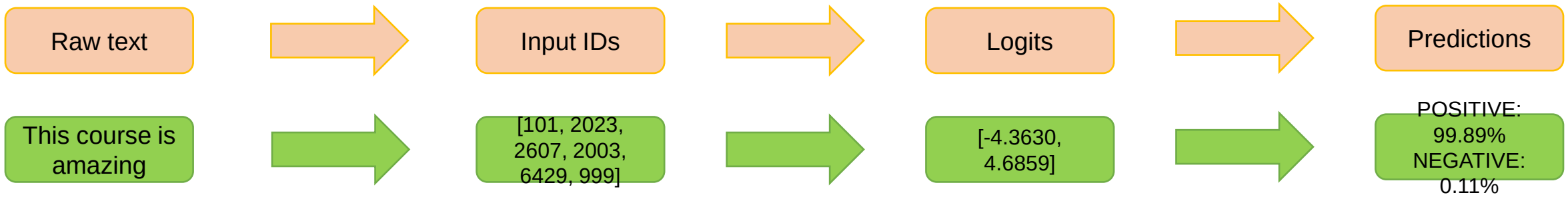
```python
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
inputs = tokenizer("Let's try to tokenize!")
print(inputs)
```

```
{
    'input_ids': [101, 2292, 1005, 1055, 3046, 2000, 19204, 4697, 999, 102],
    'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
}
```

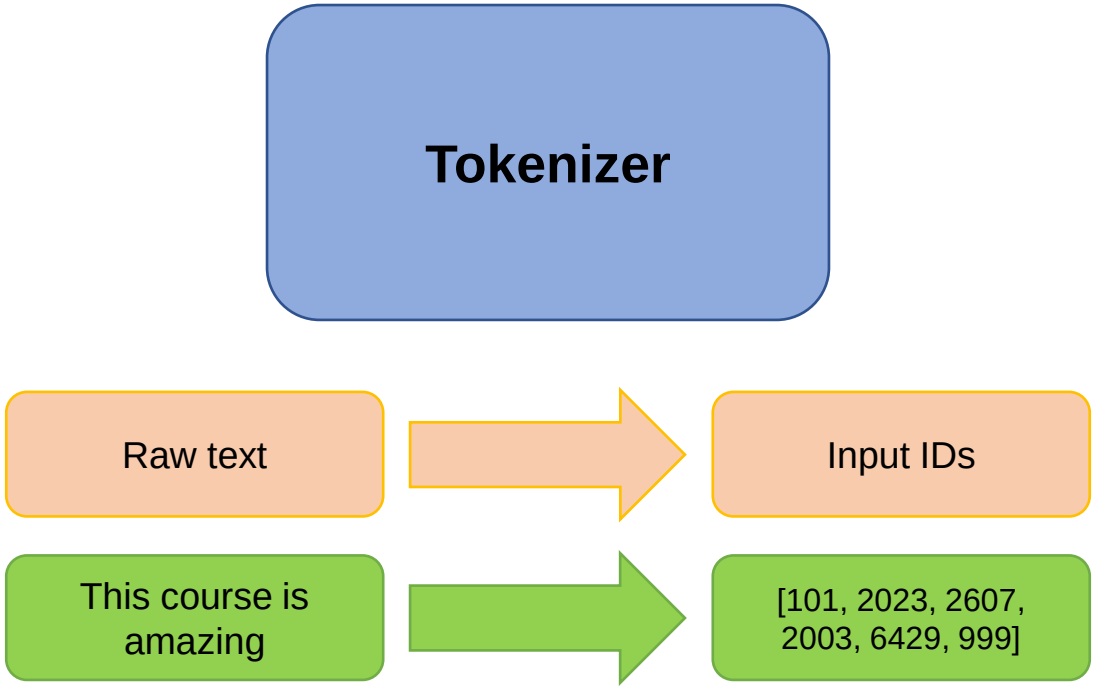
A tokenizer takes texts as inputs and outputs numbers the associated model can make sense of

What happens inside the pipeline function?

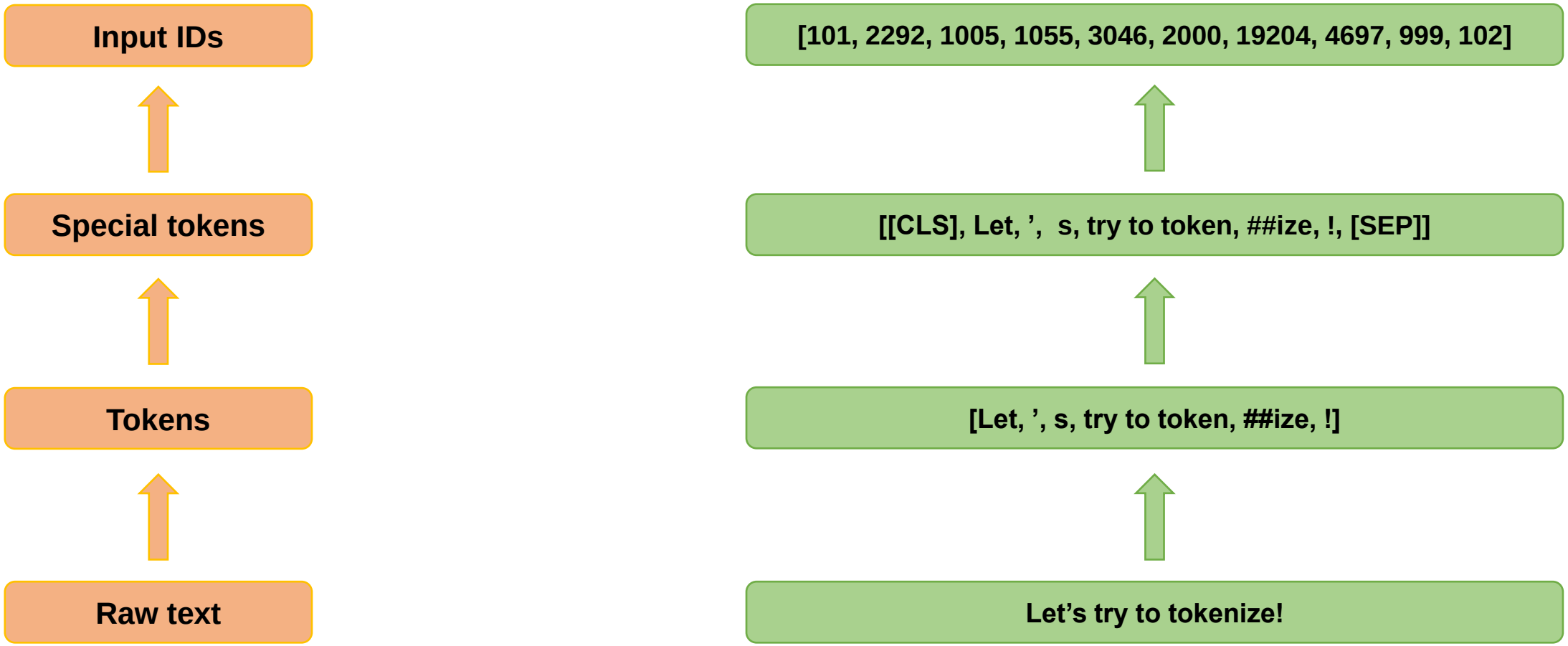




The pipeline consists of three stages



The pipeline consists of three stages

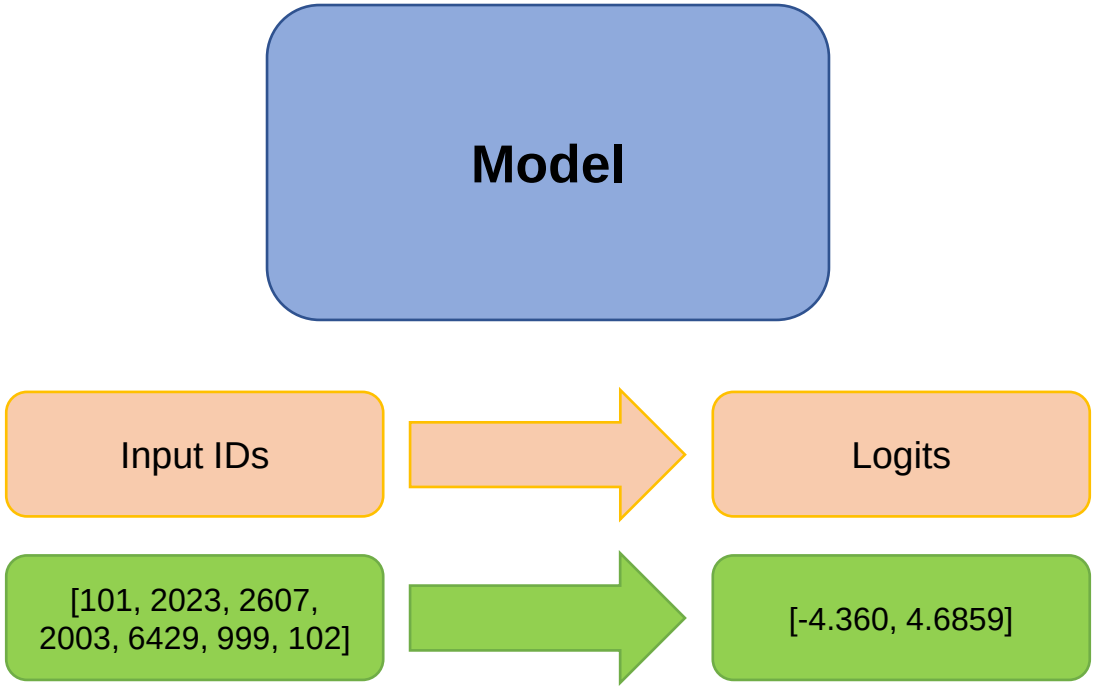


The tokenization pipeline: from input text to a list of numbers

```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]
    ])
}
```

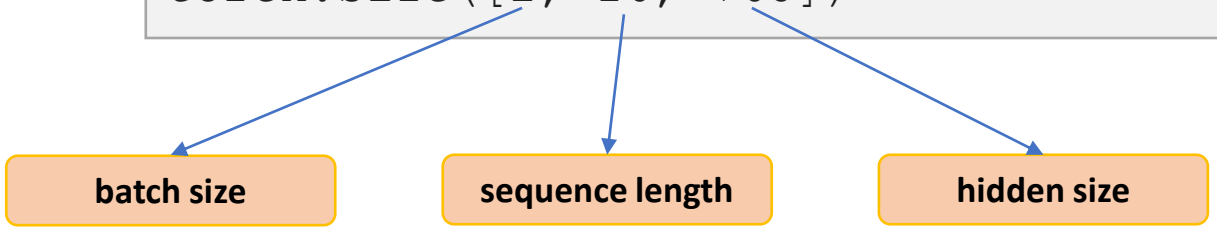
A tokenizer takes texts as inputs and outputs numbers the associated model can make sense of



Stage 2: Model

```
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])
```



The AutoModel class loads a model without its pretraining head

```
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>)
```

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

Example - Softmax

Input Image | Model | Logit (O) | Softmax | Probabilities | Label

3.2

1.3

0.2

0.8

$$\hat{y}_i = \frac{\exp(o_i)}{\sum_{j=1}^{k} \exp(o_j)}$$

Classes
Dog
Cat
Bear
Tiger

Figure Source: Dog Image

Example - Softmax

Input Image | Model | Logit (O) | Softmax | Probabilities | Label

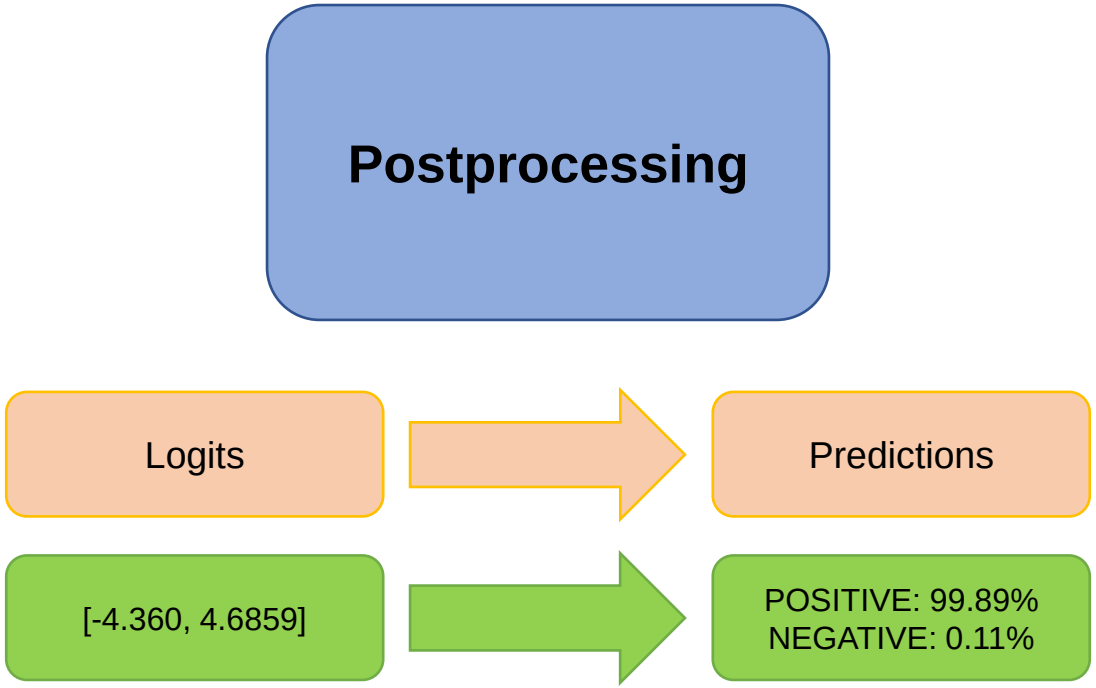
Logit (O)
3.2
1.3
0.2
0.8

$$\hat{y}_i = \frac{\exp(o_i)}{\sum_{j=1}^{k} \exp(o_j)}$$

Probabilities
0.78
0.12
0.04
0.07

Classes
Dog
Cat
Bear
Tiger

Figure Source: Dog Image



Stage 3: Postprocessing

model.config.id2label

{0: ‘NEGATIVE’,1: ‘POSITIVE’}

- First sentence: NEGATIVE 4.02%
  POSITIVE 95.98%
- Second sentence: NEGATIVE 99.46%
  POSITIVE 0.54%

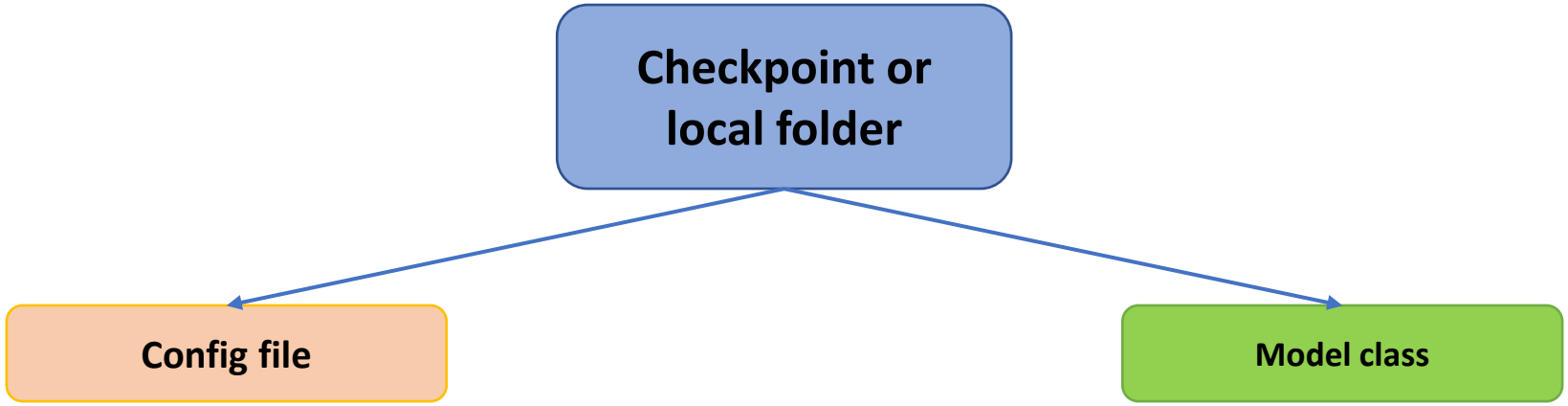
The three stages of the pipeline

Additional Content
Instantiate a Transformer model

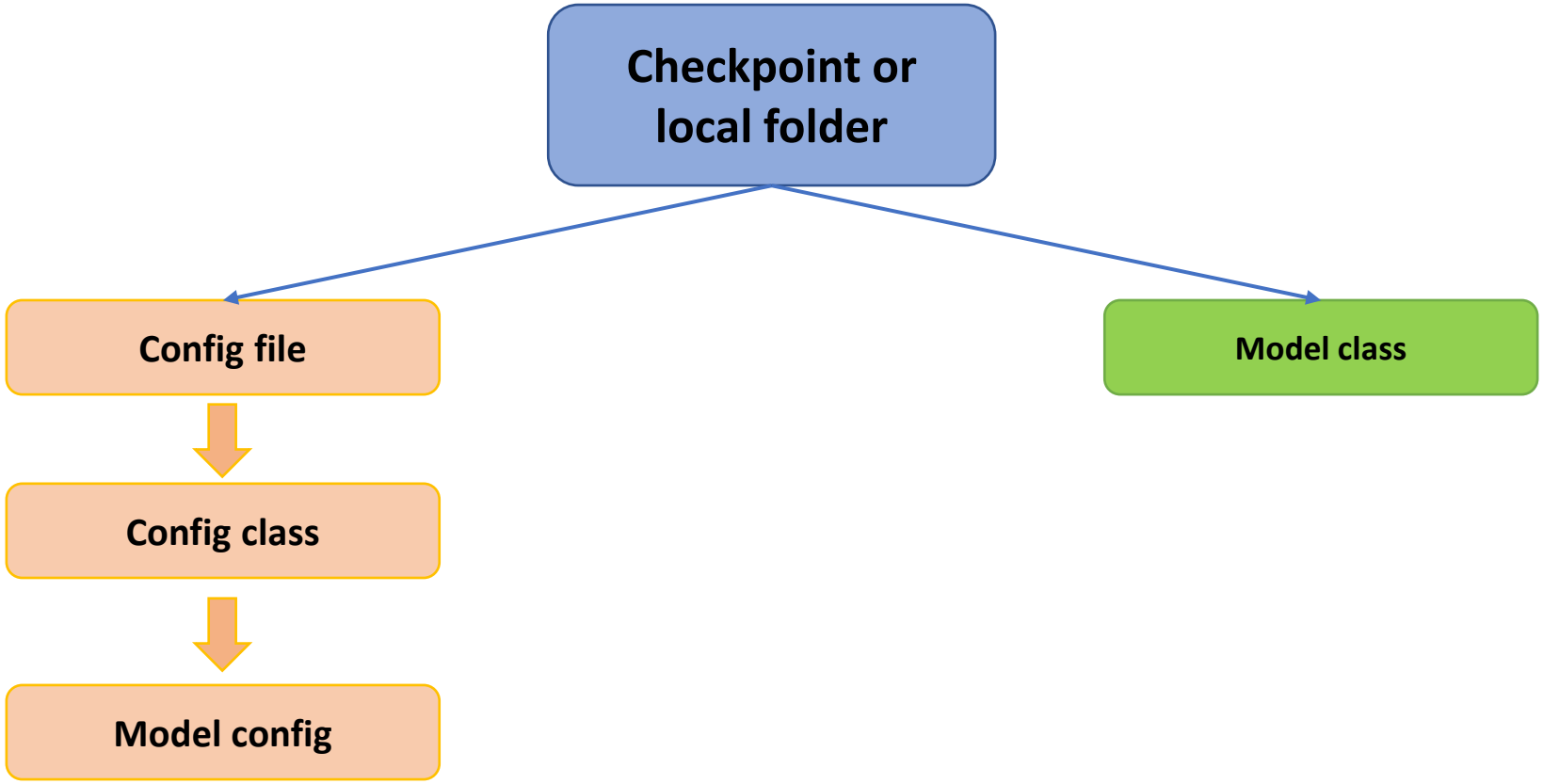
```
from transformers import AutoModel

bert_model = AutoModel.from_pretrained("bert-base-cased")
print(type(bert_model))

gpt_model = AutoModel.from_pretrained("gpt2")
print(type(gpt_model))

bart_model = AutoModel.from_pretrained("facebook/bart-base")
print(type(bart_model))
```

```
<class 'transformers.models.bert.modeling_bert.BertModel'>
<class 'transformers.models.gpt2.modeling_gpt2.GPT2Model'>
<class 'transformers.models.bart.modeling_bart.BartModel'>
```

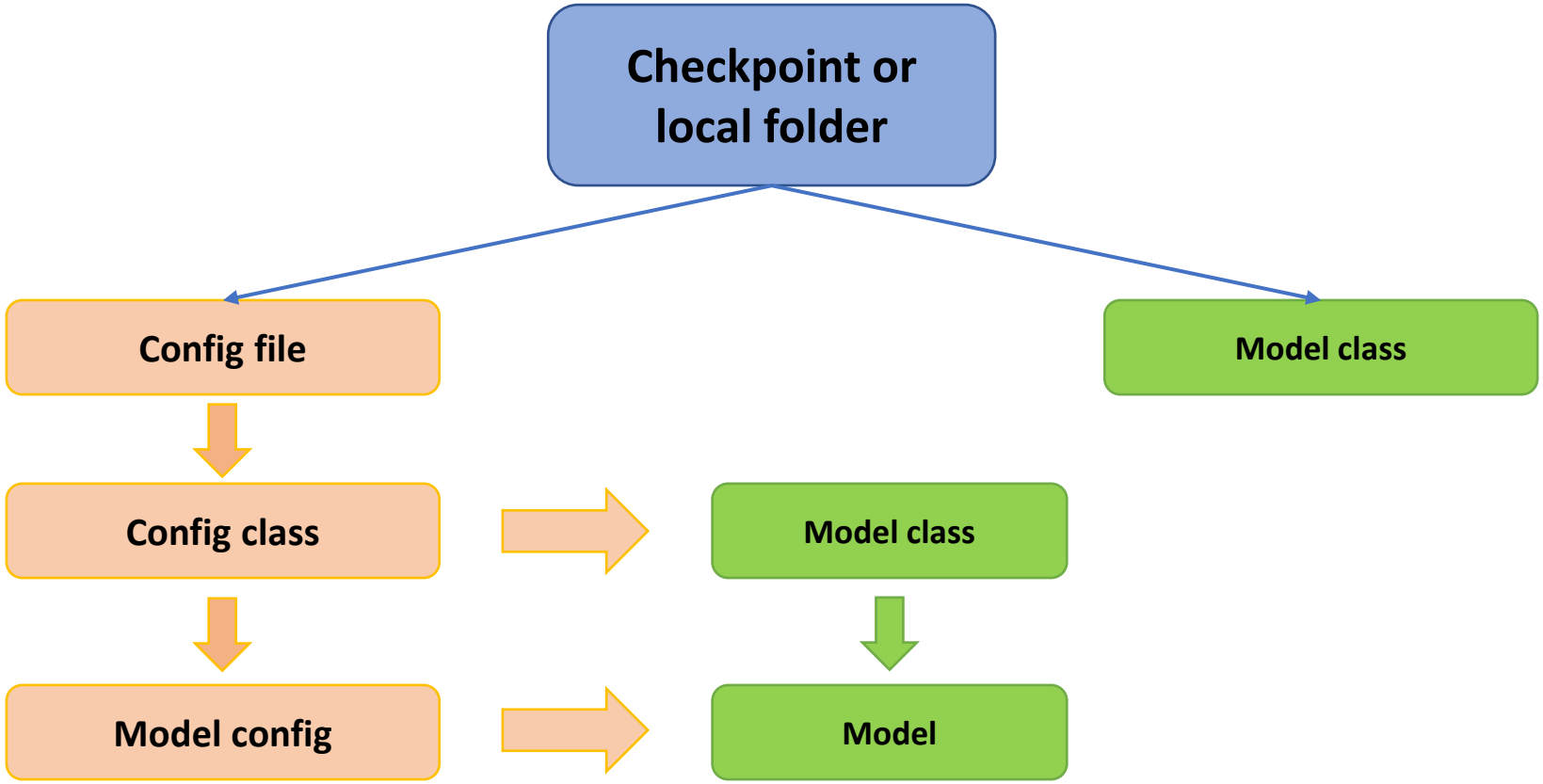
The AutoModel API allows you to instantiate a pretrained model from any checkpoint



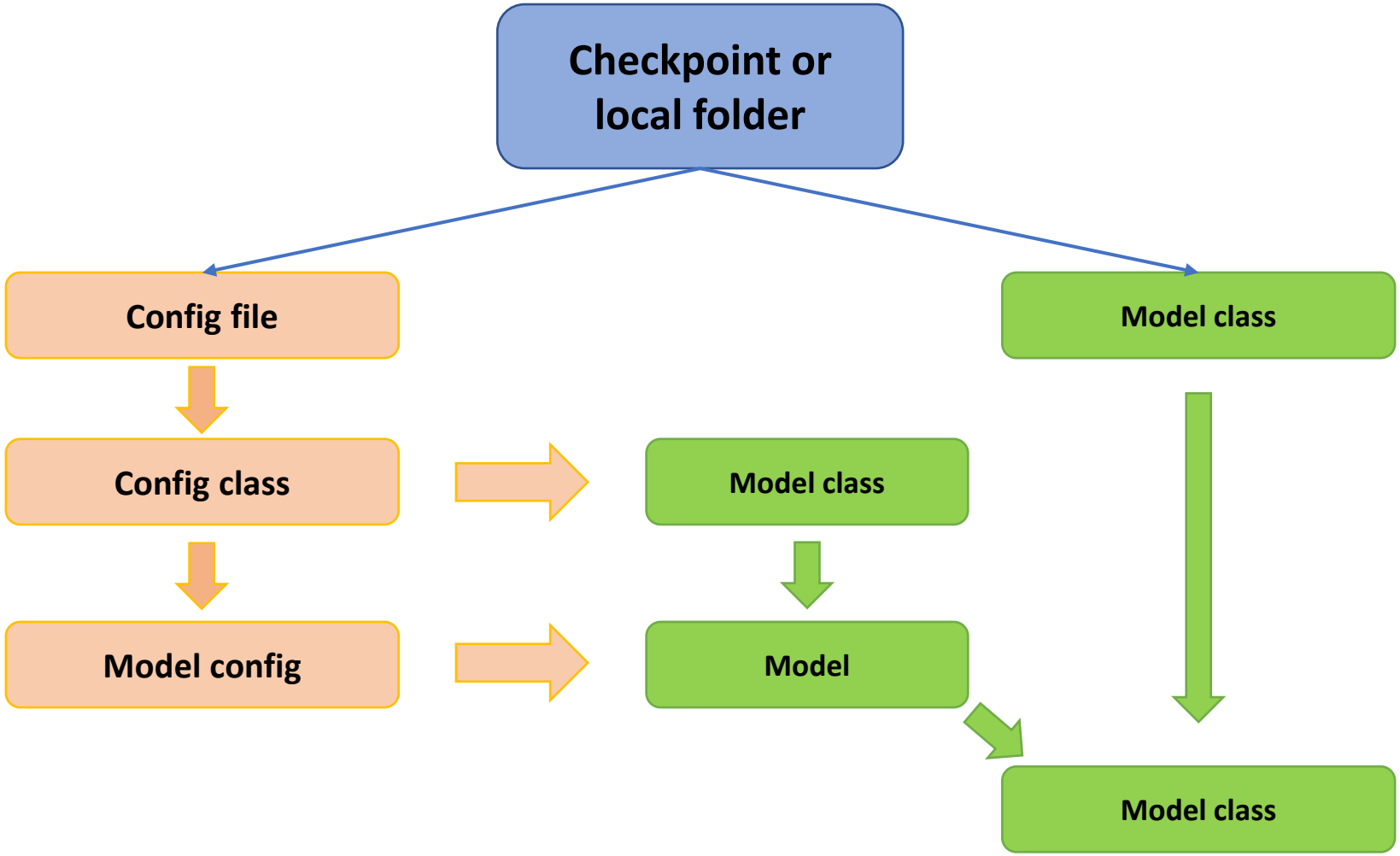
Behind the AutoModel.from_pretrained() method



Behind the AutoModel.from_pretrained() method



Behind the AutoModel.from_pretrained() method



Behind the AutoModel.from_pretrained() method

```python
from transformers import BertConfig

bert_config = BertConfig.from_pretrained("bert-base-cased")
print(type(bert_config))
```

```
<class 'transformers.models.bert.configuration_bert.BertConfig'>
```

```python
from transformers import GPT2Config

gpt_config = GPT2Config.from_pretrained("gpt2")
print(type(gpt_config))
```

```
<class 'transformers.models.gpt2.configuration_gpt2.GPT2Config'>
```

```python
from transformers import BartConfig

bart_config = BartConfig.from_pretrained("facebook/bart-base")
print(type(bart_config))
```

```
<class 'transformers.models.bart.configuration_bart.BartConfig'>
```

But you can also use the specific class if you know it

```python
from transformers import BertConfig

bert_config = BertConfig.from_pretrained("bert-base-cased")
print(bert_config)
```

```
BertConfig {
     "architectures": [
          "BertForMaskedLM"
     ],
     "attention_probs_dropout_prob": 0.1,
     "gradient_checkpointing": false,
     "hidden_act": "gelu",
     "hidden_dropout_prob": 0.1,
     "hidden_size": 768,
     "initializer_range": 0.02,
     "intermediate_size": 3072,
     "layer_norm_eps": 1e-12,
     "max_position_embeddings": 512,
     "model_type": "bert",
     "num_attention_heads": 12,
     "num_hidden_layers": 12,
     "pad_token_id": 0,
     "position_embedding_type": "absolute",
     "transformers_version": "4.7.0.dev0",
     "type_vocab_size": 2,
     "use_cache": true,
     "vocab_size": 28996
}
```

The configuration contains all the information needed to load the model

Same architecture as bert-base-cased

```
from transformers import BertConfig, BertModel

bert_config = BertConfig.from_pretrained("bert-base-cased")
bert_model = BertModel(bert_config)
```

Using only 10 layers instead of 12

```
from transformers import BertConfig, BertModel

bert_config = BertConfig.from_pretrained("bert-base-cased", num_hidden_layers=10)
bert_model = BertModel(bert_config)
```

Then you can instantiate a give model with random weights from this config

Saving a model

```
from transformers import BertConfig, BertModel

bert_config = BertConfig.from_pretrained("bert-base-cased")
bert_model = BertModel(bert_config)

# Training code

bert_model.save_pretrained("my_bert_model")
```

Reloading a saved model

```
from transfromers import BertModel

bert_model = BertModel.from_pretrained("my-bert-model")
```

To save a model we just have to use the save_pretrained method