```
from transformers import pipeline
```

```
nlp = pipeline("sentiment-analysis")
```

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision af0f99b (https://huggin
Using a pipeline without specifying a model name and revision in production is not recommended.
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as s
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json: 100% 629/629 [00:00<00:00, 12.6kB/s]
model.safetensors: 100% 268M/268M [00:03<00:00, 104MB/s]
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 936B/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 3.06MB/s]
```

```
result = nlp("Everyone hates you")
result
```

```
[{'label': 'NEGATIVE', 'score': 0.9986054301261902}]
```

```
result = nlp("Your family loves you")
result
```

```
[{'label': 'POSITIVE', 'score': 0.9998372793197632}]
```

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch
```

```
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased-finetuned-mrpc")
model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased-finetuned-mrpc")
```

```
tokenizer_config.json: 100% 49.0/49.0 [00:00<00:00, 1.92kB/s]
config.json: 100% 433/433 [00:00<00:00, 15.7kB/s]
vocab.txt: 100% 213k/213k [00:00<00:00, 10.3MB/s]
tokenizer.json: 100% 436k/436k [00:00<00:00, 21.4MB/s]
model.safetensors: 100% 433M/433M [00:04<00:00, 74.6MB/s]
```

```
classes = ["Not_Paraphrase", "Paraphrase"]
```

```
sentence1 = "A reply from you is what I'm expecting"
sentence2 = "You are very cheerful today"
sentence3 = "I am awaiting a response from you"
```

```
paraphrase = tokenizer(sentence1, sentence3, return_tensors="pt")
not_paraphrase = tokenizer(sentence1, sentence2, return_tensors="pt")
```

```
paraphrase
```

```
{'input_ids': tensor([[  101,   138,  7163,  1121,  1128,  1110,  1184,   146,   112,   182,
          7805,   102,   146,  1821, 16794,   170,  2593,  1121,  1128,   102]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]]), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
```

```
paraphrase_model = model(**paraphrase)
nonparaphrase_model = model(**not_paraphrase)
```

```
paraphrase_model, nonparaphrase_model
```

```
(SequenceClassifierOutput(loss=None, logits=tensor([[-0.1651,  1.7294]], grad_fn=<AddmmBackward0>), hidden_states=None,
attentions=None),
 SequenceClassifierOutput(loss=None, logits=tensor([[ 1.1557, -2.1164]], grad_fn=<AddmmBackward0>), hidden_states=None,
attentions=None))
```

```
paraphrase_result = torch.softmax(paraphrase_model[0], dim=1).tolist()[0]
nonparaphrase_result = torch.softmax(nonparaphrase_model[0], dim=1).tolist()[0]
```

```
paraphrase_result
```

```
[0.13073278963565826, 0.8692672252655029]
```

```
# Paraphrase output
for i in range(len(classes)):
    print(f"{classes[i]}: {paraphrase_result[i] * 100:.2f}%")
```

```
Not_Paraphrase: 13.07%
Paraphrase: 86.93%
```

```
# Non Paraphrase output
for i in range(len(classes)):
    print(f"{classes[i]}: {paraphrase_result[i] * 100:.2f}%")
```

```
Not_Paraphrase: 13.07%
Paraphrase: 86.93%
```

```
nlp = pipeline("question-answering")

context = r'''Apollo ran from 1961 to 1972, and was supported by the two-man Gemini program which ran concurrently with it from 1962
to 1966. Gemini missions developed some of the space travel techniques that were necessary for the success of the Apollo missions.
Apollo used Saturn family rockets as launch vehicles. Apollo/Saturn vehicles were also used for an Apollo Applications Program,
which consisted of Skylab, a space station that supported three manned missions in 1973–74, and the Apollo–Soyuz Test Project,
a joint Earth orbit mission with the Soviet Union in 1975'''
```

```
No model was supplied, defaulted to distilbert/distilbert-base-cased-distilled-squad and revision 626af31 (https://huggingface.co/d
Using a pipeline without specifying a model name and revision in production is not recommended.
config.json: 100% 473/473 [00:00<00:00, 23.1kB/s]
model.safetensors: 100% 261M/261M [00:02<00:00, 144MB/s]
tokenizer_config.json: 100% 29.0/29.0 [00:00<00:00, 1.33kB/s]
vocab.txt: 100% 213k/213k [00:00<00:00, 11.0MB/s]
tokenizer.json: 100% 436k/436k [00:00<00:00, 18.1MB/s]
```

```
result = nlp(question="What space station supported three manned missions in 1973–1974?", context=context)
result
```

```
{'score': 0.9971761703491211, 'start': 400, 'end': 406, 'answer': 'Skylab'}
```

```
result = nlp(question="What is Apollo–Soyuz Test Project?", context=context)
result
```

```
{'score': 0.4957573413848877,
 'start': 509,
 'end': 558,
 'answer': 'a joint Earth orbit mission with the Soviet Union'}
```

```
from transformers import AutoModelForQuestionAnswering

tokenizer = AutoTokenizer.from_pretrained("bert-large-uncased-whole-word-masking-finetuned-squad")
model = AutoModelForQuestionAnswering.from_pretrained("bert-large-uncased-whole-word-masking-finetuned-squad")
```

```
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 1.56kB/s]
config.json: 100% 443/443 [00:00<00:00, 18.5kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 8.60MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 15.7MB/s]
model.safetensors: 100% 1.34G/1.34G [00:13<00:00, 125MB/s]
Some weights of the model checkpoint at bert-large-uncased-whole-word-masking-finetuned-squad were not used when initializing BertFo
- This IS expected if you are initializing BertForQuestionAnswering from the checkpoint of a model trained on another task or with a
- This IS NOT expected if you are initializing BertForQuestionAnswering from the checkpoint of a model that you expect to be exactly
```

```
questions = ["What space station supported three manned missions in 1973–1974?", "What is Apollo–Soyuz Test Project?",
             "What are Gemini missions?"]
```

```
import torch

for question in questions:
    inputs = tokenizer(question, context, add_special_tokens=True, return_tensors="pt")
    input_ids = inputs["input_ids"].tolist()[0]

    # Pass the inputs through the model
    outputs = model(**inputs)

    # Extract answer start and end scores from the model outputs
    answer_start_scores = outputs.start_logits
    answer_end_scores = outputs.end_logits

    # Convert scores to tensors
    answer_start_scores = torch.tensor(answer_start_scores)
    answer_end_scores = torch.tensor(answer_end_scores)

    # Get the likely beginning of answer
    answer_start = torch.argmax(answer_start_scores)
    # Get the likely end of answer
    answer_end = torch.argmax(answer_end_scores) + 1

    # Convert token IDs to answer string
    answer = tokenizer.convert_tokens_to_string(tokenizer.convert_ids_to_tokens(input_ids[answer_start:answer_end]))

    print(f"Question: {question}")
    print(f"Answer: {answer}")
```

```
<ipython-input-23-51dfc3aa5d46>:15: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clone().deta
  answer_start_scores = torch.tensor(answer_start_scores)
<ipython-input-23-51dfc3aa5d46>:16: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clone().deta
  answer_end_scores = torch.tensor(answer_end_scores)
Question: What space station supported three manned missions in 1973–1974?
Answer: skylab
Question: What is Apollo–Soyuz Test Project?
Answer: a joint earth orbit mission with the soviet union
Question: What are Gemini missions?
Answer: developed some of the space travel techniques
```

```
from transformers import pipeline
nlp = pipeline("fill-mask")
```

```
No model was supplied, defaulted to distilbert/distilroberta-base and revision ec58a5b (https://huggingface.co/distilbert/distilrobe
Using a pipeline without specifying a model name and revision in production is not recommended.
config.json: 100%  480/480 [00:00<00:00, 27.3kB/s]
model.safetensors: 100%  331M/331M [00:02<00:00, 145MB/s]
Some weights of the model checkpoint at distilbert/distilroberta-base were not used when initializing RobertaForMaskedLM: ['roberta
- This IS expected if you are initializing RobertaForMaskedLM from the checkpoint of a model trained on another task or with another
- This IS NOT expected if you are initializing RobertaForMaskedLM from the checkpoint of a model that you expect to be exactly ident
tokenizer_config.json: 100%  25.0/25.0 [00:00<00:00, 1.38kB/s]
vocab.json: 100%  899k/899k [00:00<00:00, 30.0MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 14.4MB/s]
tokenizer.json: 100%  1.36M/1.36M [00:00<00:00, 44.5MB/s]
```

```
from pprint import pprint
pprint(nlp(f"Learning another {nlp.tokenizer.mask_token} is like becoming another person"))
```

```
[{'score': 0.8854474425315857,
  'sequence': 'Learning another language is like becoming another person',
  'token': 2777,
  'token_str': ' language'},
 {'score': 0.01875963993370533,
  'sequence': 'Learning another word is like becoming another person',
  'token': 2136,
  'token_str': ' word'},
 {'score': 0.013898340985178947,
  'sequence': 'Learning another name is like becoming another person',
  'token': 766,
  'token_str': ' name'},
 {'score': 0.013433171436190605,
  'sequence': 'Learning another skill is like becoming another person',
  'token': 6707,
  'token_str': ' skill'},
 {'score': 0.008241109549999237,
  'sequence': 'Learning another thing is like becoming another person',
  'token': 631,
  'token_str': ' thing'}]
```

```
pprint(nlp(f"I love Kaggle because it gives me {nlp.tokenizer.mask_token}"))
```

```
[{'score': 0.2796865403652191,
  'sequence': 'I love Kaggle because it gives me nightmares',
  'token': 31634,
  'token_str': ' nightmares'},
 {'score': 0.06478409469127655,
  'sequence': 'I love Kaggle because it gives me joy',
  'token': 5823,
  'token_str': ' joy'},
 {'score': 0.05835973843932152,
  'sequence': 'I love Kaggle because it gives me hope',
  'token': 1034,
  'token_str': ' hope'},
 {'score': 0.04237747564911842,
  'sequence': 'I love Kaggle because it gives me confidence',
  'token': 2123,
  'token_str': ' confidence'},
 {'score': 0.027025585994124413,
  'sequence': 'I love Kaggle because it gives me wings',
  'token': 11954,
  'token_str': ' wings'}]
```

```
from transformers import AutoModelWithLMHead, AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("distilbert-base-cased")
model = AutoModelWithLMHead.from_pretrained("distilbert-base-cased")

sequence = f"Masked language modeling is the task of masking tokens in a sequence with a masking token, and prompting the model to fill

input = tokenizer.encode(sequence, return_tensors="pt")
```

```
tokenizer_config.json: 100%  29.0/29.0 [00:00<00:00, 1.98kB/s]
config.json: 100%  465/465 [00:00<00:00, 19.8kB/s]
vocab.txt: 100%  213k/213k [00:00<00:00, 12.0MB/s]
tokenizer.json: 100%  436k/436k [00:00<00:00, 21.0MB/s]
/usr/local/lib/python3.10/dist-packages/transformers/models/auto/modeling_auto.py:1595: FutureWarning: The class `AutoModelWithLMHea
  warnings.warn(
model.safetensors: 100%  263M/263M [00:01<00:00, 156MB/s]
```

```
mask_token_id = torch.where(input == tokenizer.mask_token_id)[1]
model_output = model(input)[0]


mask_token_logits = model_output[0, mask_token_id, :]
print(mask_token_logits)
```

```
tensor([[-6.4004, -6.7450, -6.6433,  ..., -5.8269, -5.4157, -4.7601]],
       grad_fn=<IndexBackward0>)
```

```
top_5_tokens = torch.topk(mask_token_logits, 5, dim=1).indices[0].tolist()
for token in top_5_tokens:
    print(sequence.replace(tokenizer.mask_token, tokenizer.decode([token])))
```

```
Masked language modeling is the task of masking tokens in a sequence with a masking token, and prompting the model to fill gaps with
Masked language modeling is the task of masking tokens in a sequence with a masking token, and prompting the model to fill sequences
Masked language modeling is the task of masking tokens in a sequence with a masking token, and prompting the model to fill blocks wi
Masked language modeling is the task of masking tokens in a sequence with a masking token, and prompting the model to fill them with
Masked language modeling is the task of masking tokens in a sequence with a masking token, and prompting the model to fill spaces wi
```

```
from transformers import AutoModelWithLMHead, AutoTokenizer, top_k_top_p_filtering

from torch.nn import functional as F

tokenizer = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelWithLMHead.from_pretrained("gpt2")

sequence = f"Psychology is the study of human "

input_ids = tokenizer.encode(sequence, return_tensors="pt")
```

```
tokenizer_config.json: 100%  26.0/26.0 [00:00<00:00, 1.10kB/s]
config.json: 100%  665/665 [00:00<00:00, 24.3kB/s]
vocab.json: 100%  1.04M/1.04M [00:00<00:00, 19.0MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 21.2MB/s]
tokenizer.json: 100%  1.36M/1.36M [00:00<00:00, 34.3MB/s]
model.safetensors: 100%  548M/548M [00:04<00:00, 123MB/s]
generation_config.json: 100%  124/124 [00:00<00:00, 5.86kB/s]
```

```
model(input_ids)[0][:, -1, :]
```

```
tensor([[-67.0345, -67.8981, -71.2576,  ..., -77.2795, -75.8206, -69.4256]],
       grad_fn=<SliceBackward0>)
```

```
# get logits of last hidden state
next_token_logits = model(input_ids)[0][:, -1, :]

filtered_next_token_logits = top_k_top_p_filtering(next_token_logits, top_k=50, top_p=1.0)

probs = F.softmax(filtered_next_token_logits, dim=-1)
next_token = torch.multinomial(probs, num_samples=1)

generated = torch.cat([input_ids, next_token], dim=-1)

resulting_string = tokenizer.decode(generated.tolist()[0])

print(resulting_string)
```

```
Psychology is the study of human vern
```