Hugging Face Introduction

Hugging Face – The AI community building the future.

Hugging Face

Transformers library

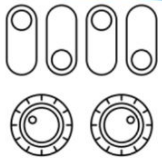
Use it with

TensorFlow

PyTorch

Use it for

Applying state of the art transformer models

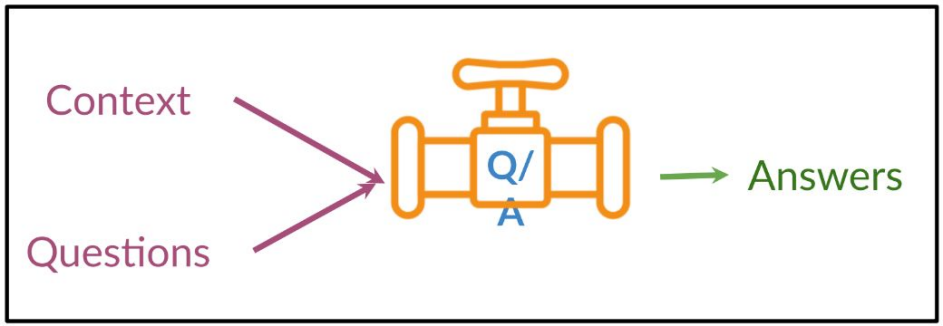


Fine-tuning pretrained transformer models

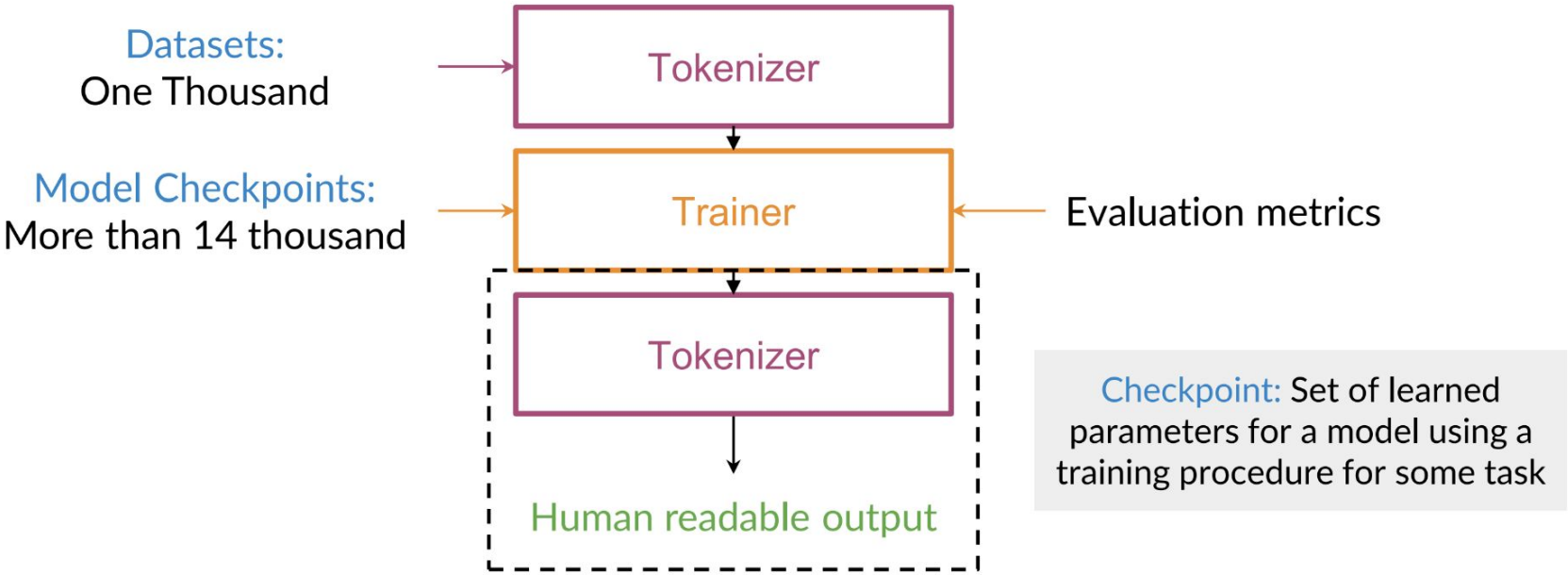
Hugging Face: Using Transformers

Pipelines

1. **Pre-processing** your inputs
2. **Running** the model
3. **Post-processing** the outputs

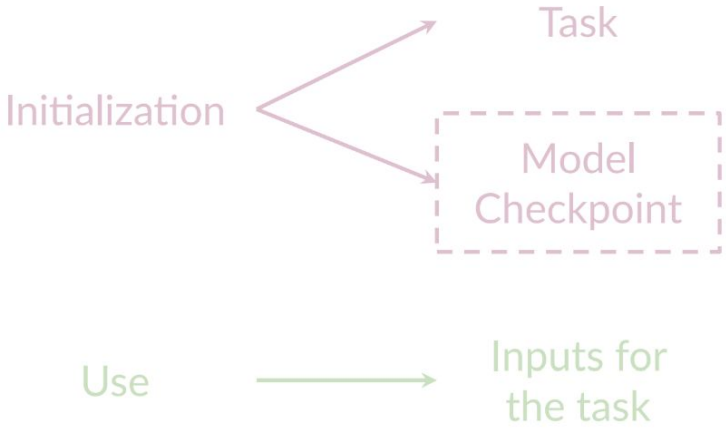


Hugging Face: Fine-Tuning Transformers



Tasks

Pipelines



Sentiment Analysis	Question Answering	Fill-Mask
Sequence	Context and questions	Sentence and position

Checkpoints

Huge number of model checkpoints that you can use in your pipelines.

But **beware**, not every checkpoint would be suitable for your task.

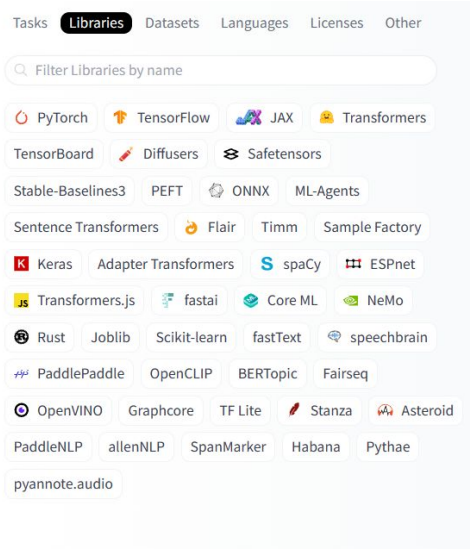
Model Checkpoints

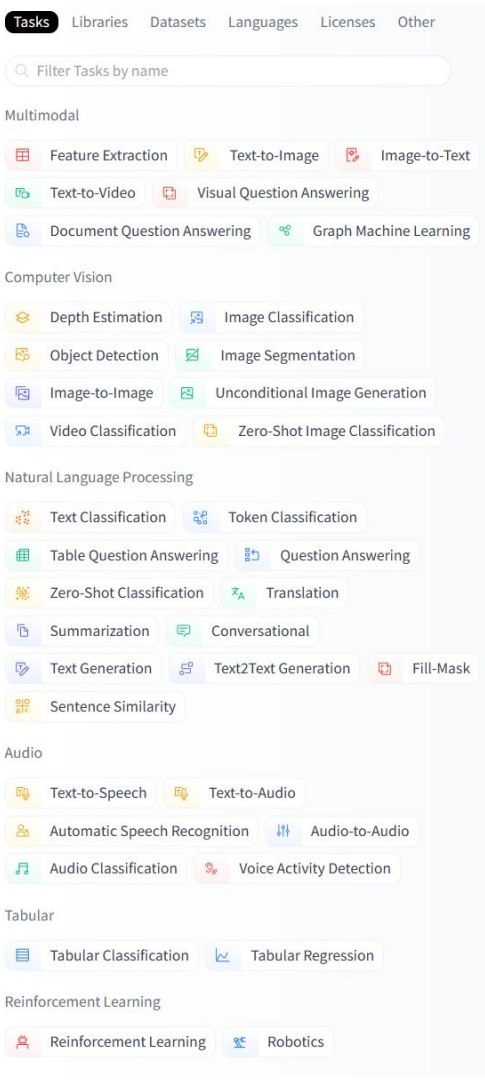
Model Checkpoints:
More than 15 thousand
(and increasing)

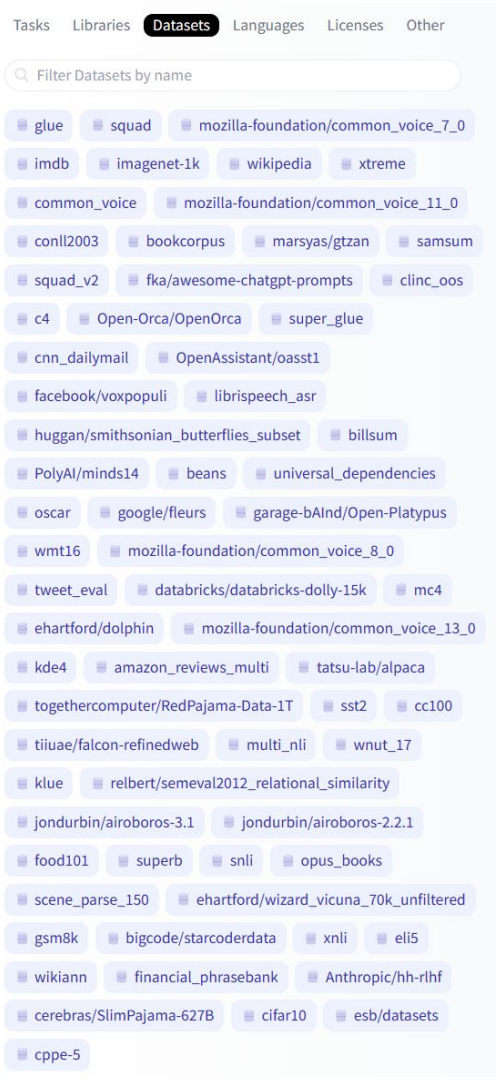
Upload the architecture and weights with 1 line of code!

Model	Dataset	Name in 🤗
DistilBERT	Stanford Question Answering Dataset (SQuAD)	distilbert-base-cased-distilled-squad
BERT	Wikipedia and Book Corpus	bert-base-cased
...	...	...

Model Card and Datasets



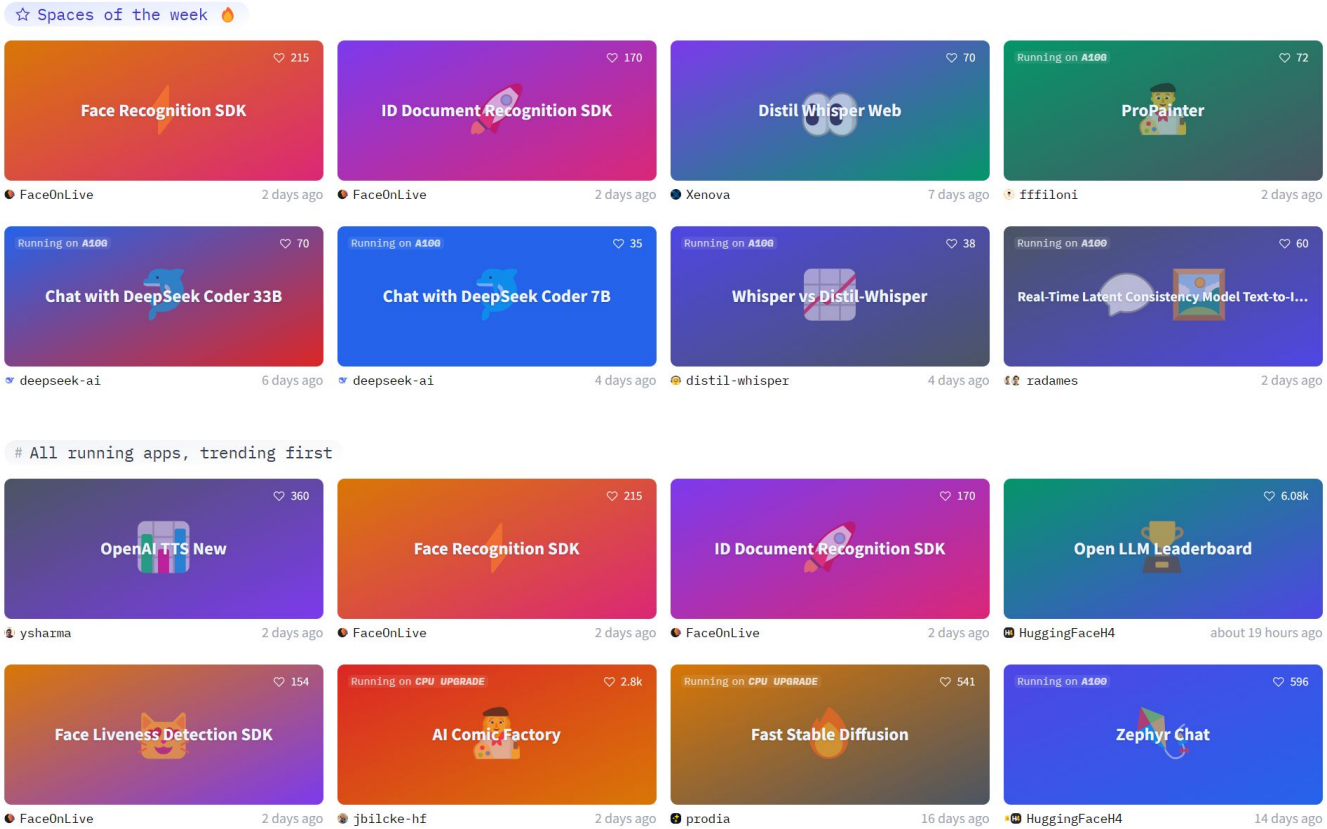




Hugging Spaces

Discover amazing ML apps made by the community!

Create new Space



Hugging Spaces APIs

Pipelines for inference

The pipeline() makes it simple to use any model from the Hub for inference on any language, computer vision, speech, and multimodal tasks. Even if you don't have experience with a specific modality or aren't familiar with the underlying code behind the models, you can still use them for inference with the pipeline()! This tutorial will teach you to:

- Use a pipeline() for inference.
- Use a specific tokenizer or model.
- Use a pipeline() for audio, vision, and multimodal tasks.

```
from transformers import pipeline

transcriber = pipeline(task= "automatic-speech-recognition" )

transcriber("https://huggingface.co/datasets/Narsil/asr_dummy/resolve/main/mlk.flac"  )

Output:

{'text': 'I HAVE A DREAM BUT ONE DAY THIS NATION WILL RISE UP LIVE UP THE TRUE MEANING OF ITS TREES'  }
```

Load pretrained instances with an AutoClass

With so many different Transformer architectures, it can be challenging to create one for your checkpoint. As a part of 🤗 Transformers core philosophy to make the library easy, simple and flexible to use, an `AutoClass` automatically infers and loads the correct architecture from a given checkpoint. The `from_pretrained()` method lets you quickly load a pretrained model for any architecture so you don't have to devote time and resources to train a model from scratch. Producing this type of checkpoint-agnostic code means if your code works for one checkpoint, it will work with another checkpoint - as long as it was trained for a similar task - even if the architecture is different.


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

sequence = "In a hole in the ground there lived a hobbit."
print(tokenizer(sequence))

Output:
{'input ids': [101, 1999, 1037, 4920, 1999, 1996, 2598, 2045, 2973, 1037, 7570,
10322, 4183, 1012, 102],
'token type ids': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

```
from transformers import AutoImageProcessor

image_processor = AutoImageProcessor.from_pretrained( "google/vit-base-patch16-224" )
```

```
from transformers import AutoFeatureExtractor

feature_extractor = AutoFeatureExtractor.from_pretrained(
...     "ehcalabres/wav2vec2-lg-xlsr-en-speech-emotion-recognition" )
```

```
from transformers import AutoProcessor

processor = AutoProcessor.from_pretrained( "microsoft/layoutlmv2-base-uncased" )
```

```
from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained( "distilbert-base-uncased" )
```

```
from transformers import AutoModelForTokenClassification

model = AutoModelForTokenClassification.from_pretrained( "distilbert-base-uncased" )
```