Huggingface Pipelines

The HuggingFace transformers library provides APIs at two different levels.

The High Level API for using open-source models for typical inference tasks is called "pipelines". It's incredibly easy to use.

You create a pipeline using something like:

```
my_pipeline = pipeline("the_task_I_want_to_do")
```

Followed by

```
result = my_pipeline(my_input)
```

And that's it!

See end of this colab for a list of all pipelines.

Before we start: 2 important pro-tips for using Colab:

Pro-tip 1:

The top of every colab has some pip installs. You may receive errors from pip when you run this, such as:

> gcsfs 2025.3.2 requires fsspec==2025.3.2, but you have fsspec 2025.3.0 which is incompatible.

These pip compatibility errors can be safely ignored; and while it's tempting to try to fix them by changing version numbers, that will actually introduce real problems!

Pro-tip 2:

In the middle of running a Colab, you might get an error like this:

> Runtime error: CUDA is required but not available for bitsandbytes. Please consider installing [...]

This is a super-misleading error message! Please don't try changing versions of packages...

This actually happens because Google has switched out your Colab runtime, perhaps because Google Colab was too busy. The solution is:

1. Kernel menu >> Disconnect and delete runtime
2. Reload the colab from fresh and Edit menu >> Clear All Outputs
3. Connect to a new T4 using the button at the top right
4. Select "View resources" from the menu on the top right to confirm you have a GPU
5. Rerun the cells in the colab, from the top down, starting with the pip installs

And all should work great - otherwise, ask me!

A sidenote:

You may already know this, but just in case you're not familiar with the word "inference" that I use here:

When working with Data Science models, you could be carrying out 2 very different activities: **training** and **inference**.

1. Training

Training is when you provide a model with data for it to adapt to get better at a task in the future. It does this by updating its internal settings - the parameters or weights of the model. If you're Training a model that's already had some training, the activity is called "fine-tuning".

2. Inference

Inference is when you are working with a model that has *already been trained*. You are using that model to produce new outputs on new inputs, taking advantage of everything it learned while it was being trained. Inference is also sometimes referred to as "Execution" or "Running a model".

All of our use of APIs for GPT, Claude and Gemini in the last weeks are examples of **inference**. The "P" in GPT stands for "Pre-trained", meaning that it has already been trained with data (lots of it!) In week 6 we will try fine-tuning GPT ourselves.

The pipelines API in HuggingFace is only for use for **inference** - running a model that has already been trained. In week 7 we will be training our own model, and we will need to use the more advanced HuggingFace APIs that we look at in the up-coming lecture.

I recorded this playlist on YouTube with more on parameters, training and inference:
https://www.youtube.com/playlist?list=PLWHe-9GP9SMMdl6SLaovUQF2abiLGbMjs

```
# if this gives an "ERROR" about pip dependency conflicts, ignore it! It doesn't affect anything.

!pip install -q -U transformers datasets diffusers
```

```
                                        491.5/491.5 kB 25.9 MB/s eta 0:00:00
                                        193.6/193.6 kB 13.2 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the sou
gcsfs 2025.3.2 requires fsspec==2025.3.2, but you have fsspec 2025.3.0 which is incompatible.
torch 2.6.0+cu124 requires nvidia-cublas-cu12==12.4.5.8; platform_system == "Linux" and platform_machine == "x86_64", but you have r
torch 2.6.0+cu124 requires nvidia-cuda-cupti-cu12==12.4.127; platform_system == "Linux" and platform_machine == "x86_64", but you ha
torch 2.6.0+cu124 requires nvidia-cuda-nvrtc-cu12==12.4.127; platform_system == "Linux" and platform_machine == "x86_64", but you ha
torch 2.6.0+cu124 requires nvidia-cuda-runtime-cu12==12.4.127; platform_system == "Linux" and platform_machine == "x86_64", but you
torch 2.6.0+cu124 requires nvidia-cudnn-cu12==9.1.0.70; platform_system == "Linux" and platform_machine == "x86_64", but you have nv
torch 2.6.0+cu124 requires nvidia-cufft-cu12==11.2.1.3; platform_system == "Linux" and platform_machine == "x86_64", but you have nv
torch 2.6.0+cu124 requires nvidia-curand-cu12==10.3.5.147; platform_system == "Linux" and platform_machine == "x86_64", but you have
torch 2.6.0+cu124 requires nvidia-cusolver-cu12==11.6.1.9; platform_system == "Linux" and platform_machine == "x86_64", but you have
torch 2.6.0+cu124 requires nvidia-cusparse-cu12==12.3.1.170; platform_system == "Linux" and platform_machine == "x86_64", but you ha
torch 2.6.0+cu124 requires nvidia-nvjitlink-cu12==12.4.127; platform_system == "Linux" and platform_machine == "x86_64", but you hav
```

```
# Imports

import torch
from google.colab import userdata
from huggingface_hub import login
from transformers import pipeline
from diffusers import DiffusionPipeline
from datasets import load_dataset
import soundfile as sf
from IPython.display import Audio
```

Important Note

I didn't mention this in the lecture, but you may need to log in to the HuggingFace hub if you've not done so before.

1. If you haven't already done so, create a **free** HuggingFace account at https://huggingface.co and navigate to Settings from the user menu on the top right. Then Create a new API token, giving yourself write permissions.

IMPORTANT when you create your HuggingFace API key, please be sure to select read and write permissions for your key by clicking on the WRITE tab, otherwise you may get problems later.

2. Back here in colab, press the "key" icon on the side panel to the left, and add a new secret:
   In the name field put `HF_TOKEN`
   In the value field put your actual token: `hf_...`
   Ensure the notebook access switch is turned ON.
3. Execute the cell below to log in. You'll need to do this on each of your colabs. It's a really useful way to manage your secrets without needing to type them into colab. There's also a shortcut to simply overwrite the line below with:
   `hf_token = "hf_...."`
   But this isn't a best practice, as you'd have to be careful not to share the colab. And one of the great things about colabs is that you can share them!

```
hf_token = userdata.get('HF_TOKEN')
login(hf_token, add_to_git_credential=True)
```

```
# Sentiment Analysis

classifier = pipeline("sentiment-analysis", device="cuda")
result = classifier("I'm super excited to be on the way to LLM mastery!")
print(result)
```

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision 714eb0f (https://huggin
Using a pipeline without specifying a model name and revision in production is not recommended.
Device set to use cuda
[{'label': 'POSITIVE', 'score': 0.9993460774421692}]
```

```
# Named Entity Recognition

ner = pipeline("ner", grouped_entities=True, device="cuda")
result = ner("Barack Obama was the 44th president of the United States.")
print(result)
```

```
No model was supplied, defaulted to dbmdz/bert-large-cased-finetuned-conll03-english and revision 4c53496 (https://huggingface.co/db
Using a pipeline without specifying a model name and revision in production is not recommended.
config.json: 100%  998/998 [00:00<00:00, 90.9kB/s]
model.safetensors: 100%  1.33G/1.33G [00:21<00:00, 60.4MB/s]
Some weights of the model checkpoint at dbmdz/bert-large-cased-finetuned-conll03-english were not used when initializing BertForToke
- This IS expected if you are initializing BertForTokenClassification from the checkpoint of a model trained on another task or with
- This IS NOT expected if you are initializing BertForTokenClassification from the checkpoint of a model that you expect to be exact
tokenizer_config.json: 100%  60.0/60.0 [00:00<00:00, 4.76kB/s]
vocab.txt: 100%  213k/213k [00:00<00:00, 16.6MB/s]
Device set to use cuda
[{'entity_group': 'PER', 'score': np.float32(0.99918306), 'word': 'Barack Obama', 'start': 0, 'end': 12}, {'entity_group': 'LOC', 's
/usr/local/lib/python3.11/dist-packages/transformers/pipelines/token_classification.py:170: UserWarning: `grouped_entities` is depre
  warnings.warn(
```

```
# Question Answering with Context

question_answerer = pipeline("question-answering", device="cuda")
result = question_answerer(question="Who was the 44th president of the United States?", context="Barack Obama was the 44th president of t
print(result)
```

```
No model was supplied, defaulted to distilbert/distilbert-base-cased-distilled-squad and revision 564e9b5 (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, 49.9kB/s]
model.safetensors: 100%  261M/261M [00:04<00:00, 78.8MB/s]
tokenizer_config.json: 100%  49.0/49.0 [00:00<00:00, 2.18kB/s]
vocab.txt: 100%  213k/213k [00:00<00:00, 20.5MB/s]
tokenizer.json: 100%  436k/436k [00:00<00:00, 18.4MB/s]
Device set to use cuda
{'score': 0.9889456033706665, 'start': 0, 'end': 12, 'answer': 'Barack Obama'}
```

```
# Text Summarization

summarizer = pipeline("summarization", device="cuda")
text = """The Hugging Face transformers library is an incredibly versatile and powerful tool for natural language processing (NLP).
It allows users to perform a wide range of tasks such as text classification, named entity recognition, and question answering, among ot
It's an extremely popular library that's widely used by the open-source data science community.
It lowers the barrier to entry into the field by providing Data Scientists with a productive, convenient way to work with transformer mo
"""
summary = summarizer(text, max_length=50, min_length=25, do_sample=False)
print(summary[0]['summary_text'])
```

```
No model was supplied, defaulted to sshleifer/distilbart-cnn-12-6 and revision a4f8f3e (https://huggingface.co/sshleifer/distilbart
Using a pipeline without specifying a model name and revision in production is not recommended.
config.json: 100%  1.80k/1.80k [00:00<00:00, 122kB/s]
pytorch_model.bin: 100%  1.22G/1.22G [00:04<00:00, 199MB/s]
model.safetensors: 100%  1.22G/1.22G [00:06<00:00, 129MB/s]
tokenizer_config.json: 100%  26.0/26.0 [00:00<00:00, 1.03kB/s]
vocab.json: 100%  899k/899k [00:00<00:00, 54.6MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 1.07MB/s]
Device set to use cuda
 The Hugging Face transformers library is an incredibly versatile and powerful tool for natural language processing . It allows user
```

```
# Translation

translator = pipeline("translation_en_to_fr", device="cuda")
result = translator("The Data Scientists were truly amazed by the power and simplicity of the HuggingFace pipeline API.")
print(result[0]['translation_text'])
```

```
No model was supplied, defaulted to google-t5/t5-base and revision a9723ea (https://huggingface.co/google-t5/t5-base).
Using a pipeline without specifying a model name and revision in production is not recommended.
```

config.json: 100% 1.21k/1.21k [00:00<00:00, 124kB/s]

model.safetensors: 100% 892M/892M [00:12<00:00, 125MB/s]

generation_config.json: 100% 147/147 [00:00<00:00, 16.2kB/s]

spiece.model: 100% 792k/792k [00:00<00:00, 53.7MB/s]

tokenizer.json: 100% 1.39M/1.39M [00:00<00:00, 6.30MB/s]

```
Device set to use cuda
Les Data Scientists ont été vraiment étonnés par la puissance et la simplicité de l'API du pipeline HuggingFace.
```

```
# Another translation, showing a model being specified
# All translation models are here: https://huggingface.co/models?pipeline_tag=translation&sort=trending

translator = pipeline("translation_en_to_es", model="Helsinki-NLP/opus-mt-en-es", device="cuda")
result = translator("The Data Scientists were truly amazed by the power and simplicity of the HuggingFace pipeline API.")
print(result[0]['translation_text'])
```

config.json: 100% 1.47k/1.47k [00:00<00:00, 115kB/s]

pytorch_model.bin: 100% 312M/312M [00:03<00:00, 94.5MB/s]

generation_config.json: 100% 293/293 [00:00<00:00, 28.4kB/s]

model.safetensors: 100% 312M/312M [00:00<00:00, 319MB/s]

tokenizer_config.json: 100% 44.0/44.0 [00:00<00:00, 3.45kB/s]

source.spm: 100% 802k/802k [00:00<00:00, 1.24MB/s]

target.spm: 100% 826k/826k [00:00<00:00, 61.6MB/s]

vocab.json: 100% 1.59M/1.59M [00:00<00:00, 3.57MB/s]

```
/usr/local/lib/python3.11/dist-packages/transformers/models/marian/tokenization_marian.py:177: UserWarning: Recommended: pip install
  warnings.warn("Recommended: pip install sacremoses.")
Device set to use cuda
Los científicos de datos estaban verdaderamente sorprendidos por el poder y la simplicidad de la API de tuberías HuggingFace.
```

```
# Classification

classifier = pipeline("zero-shot-classification", device="cuda")
result = classifier("Hugging Face's Transformers library is amazing!", candidate_labels=["technology", "sports", "politics"])
print(result)
```

```
No model was supplied, defaulted to facebook/bart-large-mnli and revision d7645e1 (https://huggingface.co/facebook/bart-large-mnli)
Using a pipeline without specifying a model name and revision in production is not recommended.
```

config.json: 100% 1.15k/1.15k [00:00<00:00, 52.4kB/s]

model.safetensors: 100% 1.63G/1.63G [00:24<00:00, 72.3MB/s]

tokenizer_config.json: 100% 26.0/26.0 [00:00<00:00, 1.92kB/s]

vocab.json: 100% 899k/899k [00:00<00:00, 4.05MB/s]

merges.txt: 100% 456k/456k [00:00<00:00, 35.2MB/s]

tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 69.5MB/s]

```
Device set to use cuda
{'sequence': "Hugging Face's Transformers library is amazing!", 'labels': ['technology', 'sports', 'politics'], 'scores': [0.949383
```

```
# Text Generation

generator = pipeline("text-generation", device="cuda")
result = generator("If there's one thing I want you to remember about using HuggingFace pipelines, it's")
print(result[0]['generated_text'])
```

```
No model was supplied, defaulted to openai-community/gpt2 and revision 607a30d (https://huggingface.co/openai-community/gpt2).
Using a pipeline without specifying a model name and revision in production is not recommended.
```

config.json: 100% 665/665 [00:00<00:00, 67.2kB/s]

model.safetensors: 100% 548M/548M [00:10<00:00, 50.8MB/s]

generation_config.json: 100% 124/124 [00:00<00:00, 10.7kB/s]

tokenizer_config.json: 100% 26.0/26.0 [00:00<00:00, 2.72kB/s]

vocab.json: 100% 1.04M/1.04M [00:00<00:00, 28.2MB/s]

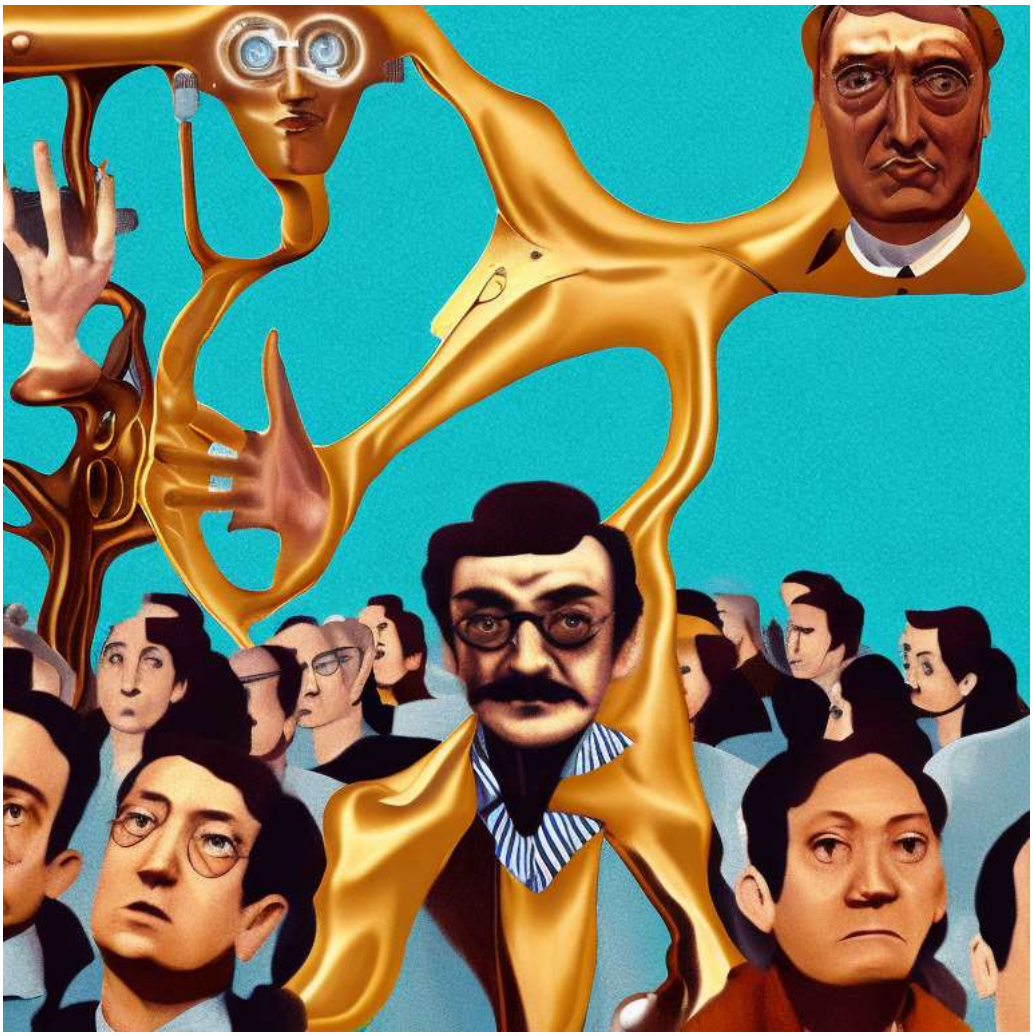
merges.txt: 100% 456k/456k [00:00<00:00, 34.3MB/s]

tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 51.5MB/s]

```
Device set to use cuda
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
If there's one thing I want you to remember about using HuggingFace pipelines, it's that you don't have to write the entire process

(g:g "fetch all the cards" ) (g:g "fetch all the cards" )

The fetching pipeline will look like this.

(g:g fetch all the cards) (g:g fetch all the cards)

With the fetching pipeline out, you can actually define a function to return a list of cards.

(g:g fetch all the cards)

Now, you can see in the following code that we have an implementation of the fetching pipeline.

(g:g fetch all the cards) (g:g fetch all the cards)

Now in your code, you can use this function when you fetch from the queue.

(g:g fetch all the cards) (g:g fetch all the cards)

Now, when you are doing something like this, you can use this function to fetch the card in your queue and you will get the list of

(g:g fetch all the cards) (g:g fetch all the cards)

And that is the very basic way of doing it.
```

```
# Image Generation

image_gen = DiffusionPipeline.from_pretrained(
    "stabilityai/stable-diffusion-2",
    torch_dtype=torch.float16,
    use_safetensors=True,
    variant="fp16"
    ).to("cuda")

text = "A class of Data Scientists learning about AI, in the surreal style of Salvador Dali"
image = image_gen(prompt=text).images[0]
image
```

```
model_index.json: 100%                                                537/537 [00:00<00:00, 59.5kB/s]
Fetching 13 files: 100%                                               13/13 [00:16<00:00,  1.49s/it]
model.fp16.safetensors: 100%                                          681M/681M [00:09<00:00, 36.9MB/s]
preprocessor_config.json: 100%                                        342/342 [00:00<00:00, 12.4kB/s]
special_tokens_map.json: 100%                                         460/460 [00:00<00:00, 10.2kB/s]
vocab.json: 100%                                                      1.06M/1.06M [00:00<00:00, 2.43MB/s]
config.json: 100%                                                     633/633 [00:00<00:00, 10.3kB/s]
tokenizer_config.json: 100%                                           824/824 [00:00<00:00, 11.4kB/s]
scheduler_config.json: 100%                                           345/345 [00:00<00:00, 6.37kB/s]
merges.txt: 100%                                                      525k/525k [00:00<00:00, 7.15MB/s]
diffusion_pytorch_model.fp16.safetensors: 100%                        167M/167M [00:01<00:00, 99.2MB/s]
diffusion_pytorch_model.fp16.safetensors: 100%                        1.73G/1.73G [00:15<00:00, 242MB/s]
config.json: 100%                                                     909/909 [00:00<00:00, 29.2kB/s]
config.json: 100%                                                     611/611 [00:00<00:00, 25.1kB/s]
Loading pipeline components...: 100%                                  6/6 [00:01<00:00,  3.25it/s]
100%                                                                  50/50 [00:18<00:00,  2.72it/s]
```



```
# Audio Generation

synthesiser = pipeline("text-to-speech", "microsoft/speecht5_tts", device='cuda')

embeddings_dataset = load_dataset("Matthijs/cmu-arctic-xvectors", split="validation")
speaker_embedding = torch.tensor(embeddings_dataset[7306]["xvector"]).unsqueeze(0)

speech = synthesiser("Hi to an artificial intelligence engineer, on the way to mastery!", forward_params={"speaker_embeddings": speaker_

sf.write("speech.wav", speech["audio"], samplerate=speech["sampling_rate"])
Audio("speech.wav")
```