Intro to deep learning

Dr. Janoś Gabler, University of Bonn

Lecture 3: First experience with HuggingFace

What you will learn in this lecture

- How to use pre-trained models from huggingface
- How te clear the model cache to free up space
- How to set some tuning parameters to get what you want
- How to ask a model about the course logistics!

Ken Judd in Bonn

- OSE meetup, Thursday, 6 pm, no registration needed
- Office hours, Thursday afternoon, register [here](IAME lounge (Lennestr. 43))
- Location of both: IAME lounge (Lennestr. 43)

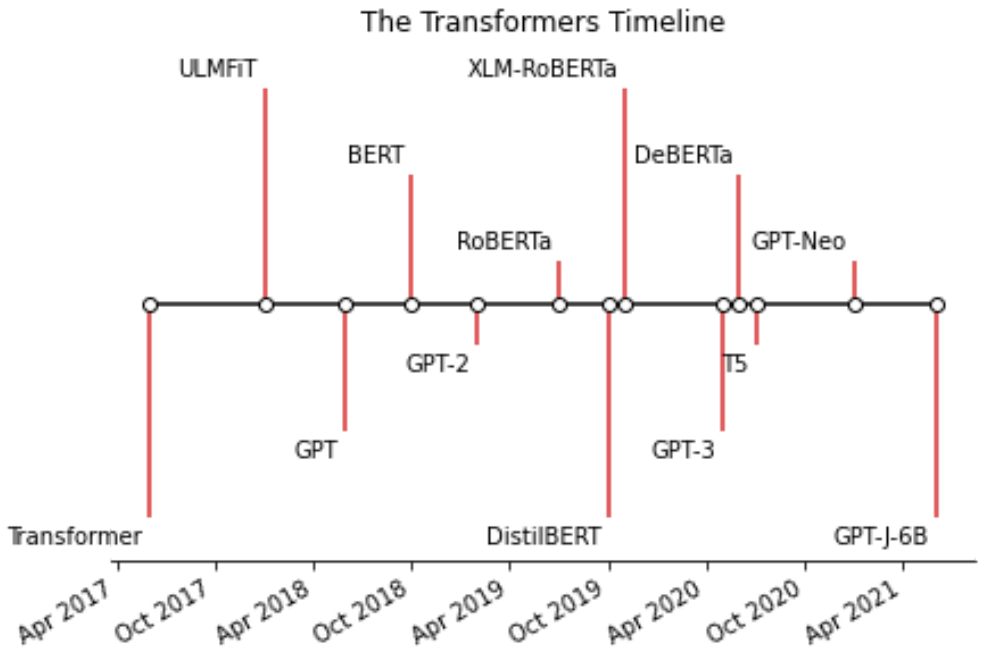
Should we start a bit earlier?

- Some students cannot attend second half
- Currently:
  - Start at 12:15
  - Break from 13:45 to 14:15
- Proposal
  - Start at 12:00
  - Have break from 14:00 to 14:30

A bit of history

- 2010: Start of deep learning revolution
  - CNNs win computer vision contests
  - Training on GPUs becomes mainstream
- Ca. 2015:
  - Transfer learning becomes mainstream in computer vision
  - DNNs outperform everything else in computer vision
- 2017 revolution starts in NLP:
  - Attention is all you need
  - ULMFiT brings transfer learning to NLP

Timeline


The Transformers Timeline
ULMFiT
XLM-RoBERTa
BERT
DeBERTa
RoBERTa
GPT-Neo
GPT-2
T5
GPT
GPT-3
Transformer
DistilBERT
GPT-J-6B
Apr 2017
Oct 2017
Apr 2018
Oct 2018
Apr 2019
Oct 2019
Apr 2020
Oct 2020
Apr 2021

What is transfer learning

- Pre-training: Train a NN on giant amounts of **unlabeled data**
  - Language modelling: Predict next word in a sequence (e.g. GPT)
  - Masked language modelling: Predict a word that was removed (e.g. Bert)
- Domain adaptation: Continue on specialized text corpus
- Fine tuning:
  - Specialize last layer to the target task
  - Train it on **labeled data**

Plan for today

- Work with pre-trained and pre-fine-tuned models!
- Meet the most relevant NLP tasks:
  - Text classification / sentiment analysis
  - Named entity recognition
  - Question answering
  - Summarization
  - Translation
- Work with the transformers library for the first time

`transformers.pipeline`

- For us: Model is a black box that takes text and returns text
- In reality:
  - Text is converted to numbers (tokenizer)
  - Main model
  - Task specific head
  - Result is converted to text again
  - Post-processing
- `transformers.pipeline` hides all of this!

Important

- All the tasks ask you to use specific models instead of default
- This is to limit downloaded file-sizes

An example text

```
text = """Dear Amazon, last week I ordered an Optimus
Prime action figure from your online store in Germany.
Unfortunately, when I opened the package, I discovered
to my horror that I had been sent an action figure of
Megatron instead! As a lifelong enemy of the Decepticons,
I hope you can understand my dilemma. To resolve the
issue, I demand an exchange of Megatron for the Optimus
Prime figure I ordered. Enclosed are copies of my records
concerning this purchase. I expect to hear from you soon.
Sincerely, Bumblebee."""
```

- Does the text have a negative or positive connotation?
- What people/objects are named in the text?
- What are main points of the text?
- What if we need a translation of the text?

Text classification

```
>>> from transformers import pipeline
>>> classifier = pipeline(task="text-classification")
>>> sentiments = classifier(text)
>>> sentiments
[{'label': 'NEGATIVE', 'score': 0.9015460014343262}]
```

- Create a `pipeline` with the task `"text-classification"`
- A task-specific default model is used, but you can define your model!
- The results are a lists of dictionaries
- Contain predictions and confidence scores.

Task 1

15 minutes

Named entity recognition

```
>>> ner_tagger = pipeline(task="ner")
>>> entities = ner_tagger(text)
>>> entities
[{'entity_group': 'ORG',
  'score': 0.9411546,
  'word': 'amazon',
  'start': 5,
  'end': 11},
 {'entity_group': 'MISC',
  'score': 0.9315696,
  'word': 'optimus prime',
  'start': 36,
  'end': 49},
  ...
]
```

- The task key is `"ner"`
- Additional arguments to define the pipeline
- The output is a list of dictionaries with recognized entities
- Entities are also classified as
  - Organizations
  - People
  - ...

Task 2

10 min

Optional arguments

```
>>> def add(x, y=2):
...     return x + y
```

```
>>> add(x=4, y=5)
9
```

```
>>> add(x=4)
6
```

```
>>> add(y=4)
---------------------------------------------------------------
TypeError                 Traceback (most recent call la
Cell In[15], line 4
      1 def add(x, y=2):
      2     return x + y
----> 4 add(y=4)

TypeError: add() missing 1 required positional argument: '
```

- Python functions can have optional arguments
- Use `=` sign in function signature to define them
- The default can be overwritten in function calls

Optional arguments in pipeline

- Which optional arguments can be used, depends on the model / task
- Sometimes a bit hard to find in the documentation
- For the NER task, we look at the argument `aggregation_strategy`

(Default: `simple`). There are several aggregation strategies:

`none`: Every token gets classified without further aggregation.

`simple`: Entities are grouped according to the default schema (B-, I- tags get merged when the tag is similar).

`first`: Same as the `simple` strategy except words cannot end up with different tags. Words will use the tag of the first token when there is ambiguity.

`average`: Same as the `simple` strategy except words cannot end up with different tags. Scores are averaged across tokens and then the maximum label is applied.

`max`: Same as the `simple` strategy except words cannot end up with different tags. Word entity will be the token with the maximum score.

Pro-tip

- Always read the documentation of the libraries you are using
- Knowing and using optional arguments can set you apart from most others!
- Default values are not always chosen well
- Some arguments have a default value but should not

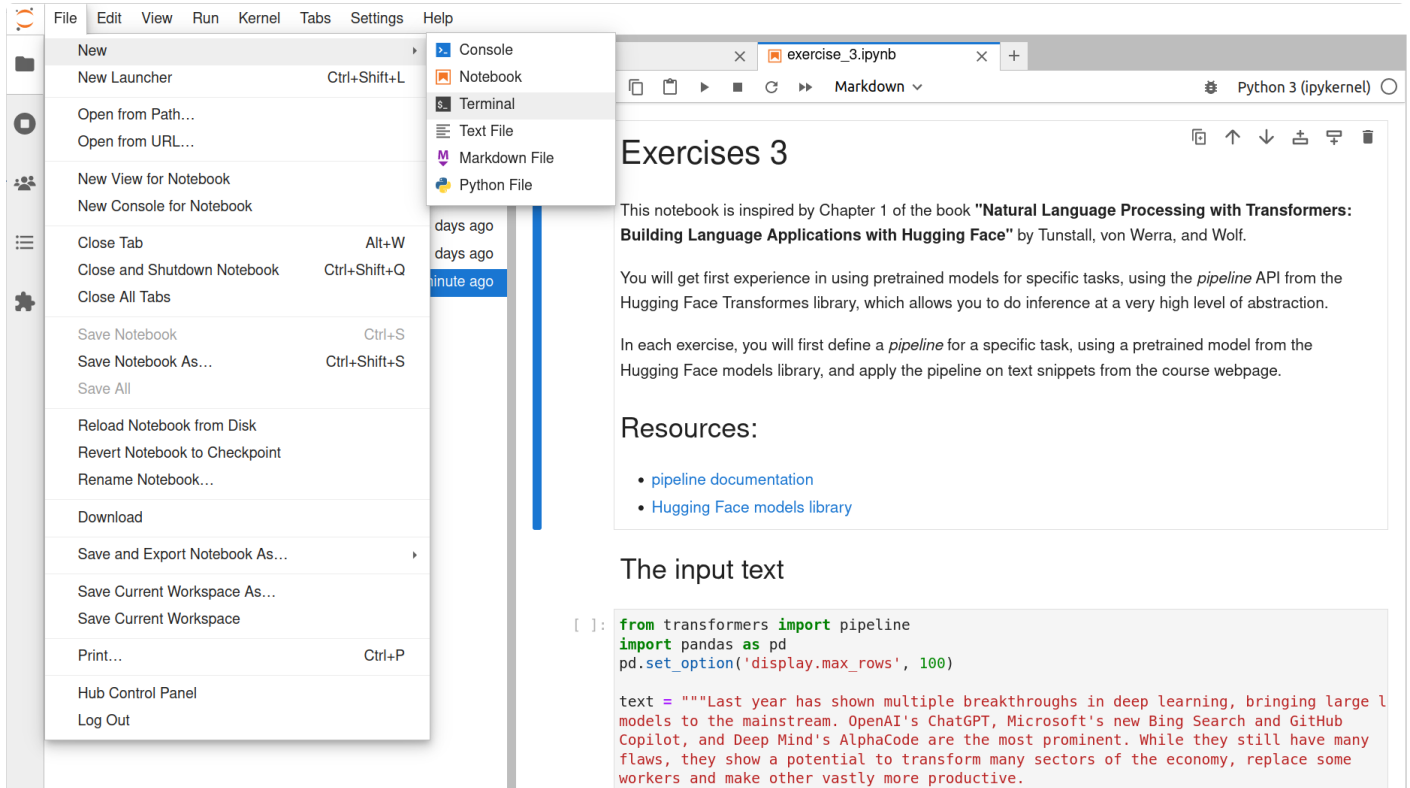
Task 3

(10 min)

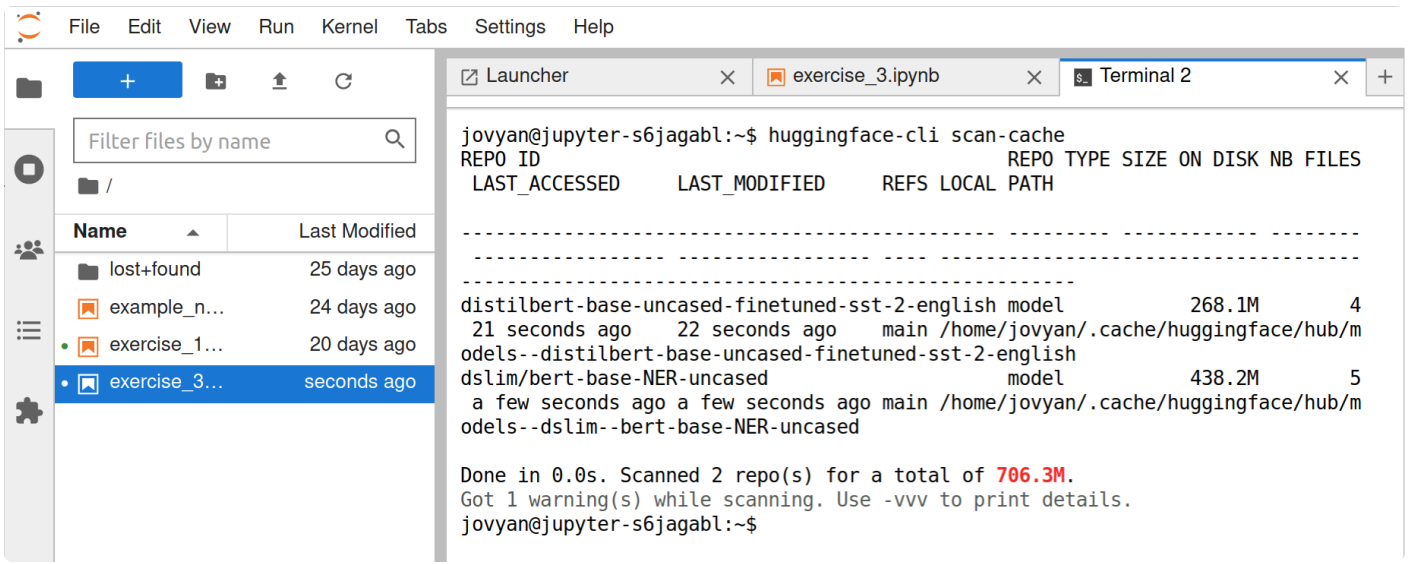
Clearing the cache

- Models are large (200 MP to several GB)
- On JupyterHub you only have 3 GB of space
- Even on your laptop, you might run out of space
- Need to learn how to delete models
- Similar on all platforms
- Important: On your laptop you need to activate the environment!

Open a terminal on JupyterHub



List your models with `scan-cache`



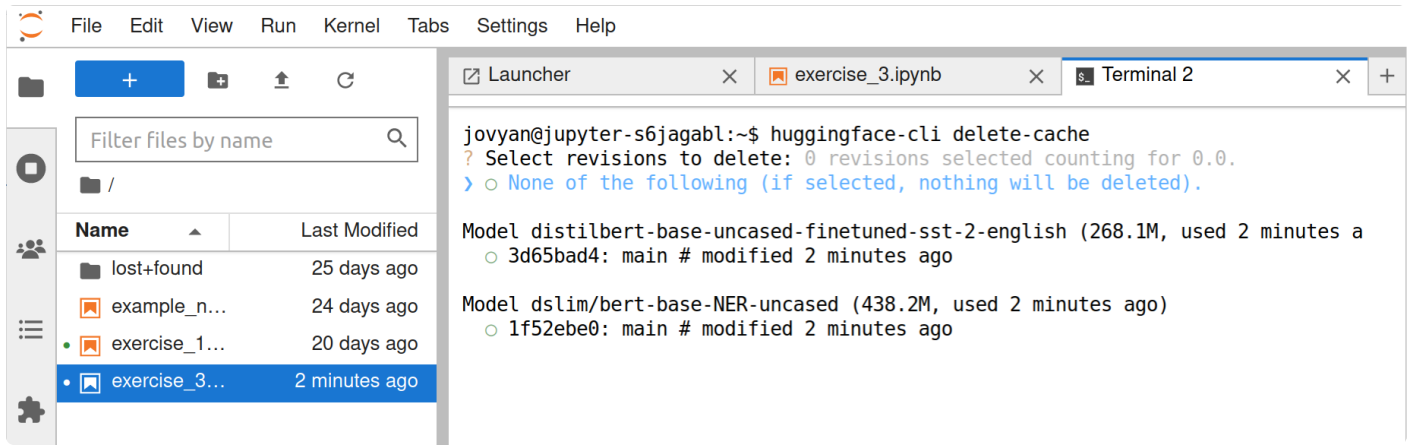
If you forgot to activate the environment

```
~
❯ huggingface-cli scan-cache
huggingface-cli: command not found

~
❯
```

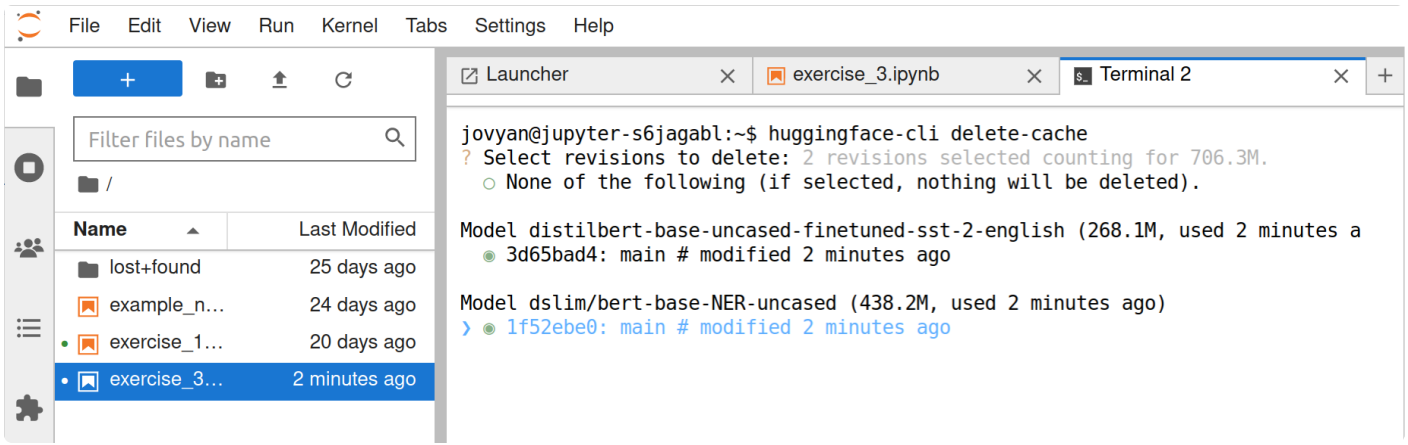
- Read the installation guide for more information

Deleting the cache: command



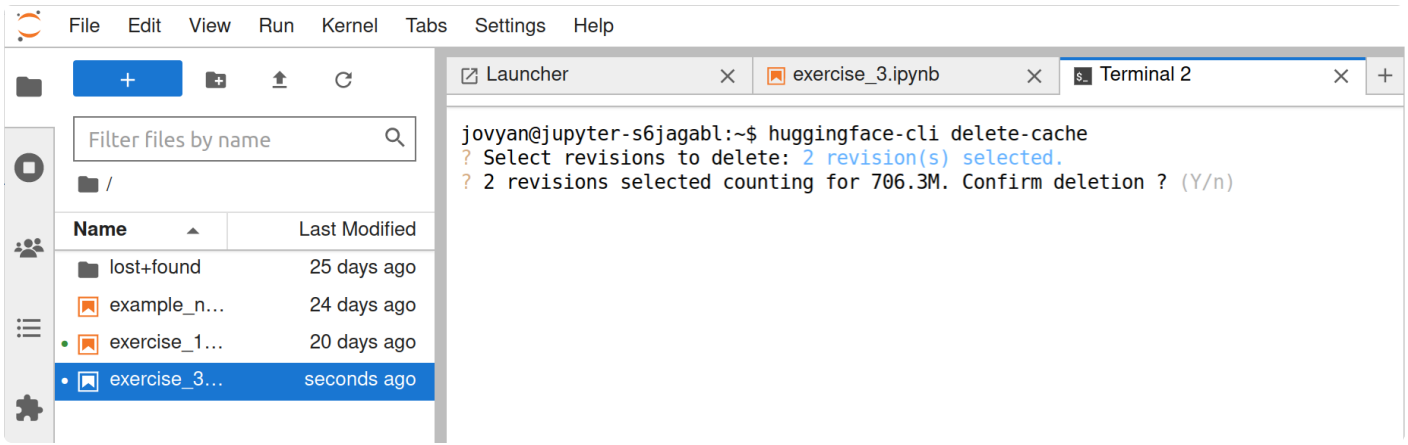
- Use the arrow keys to move up and down
- Use the space key to toggle a selection

Deleting the cache: selection



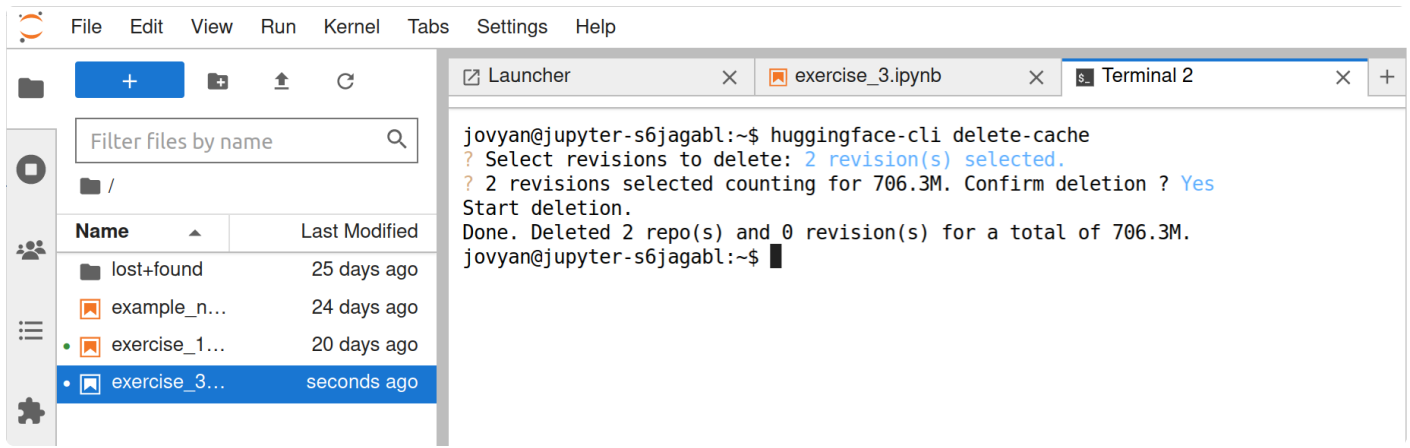
- press enter

Deleting the cache: confirmation



- confirm by typing y

Deleting the cache: done



Task 4

7 min

Question answering

```
>>> reader = pipeline(task="question-answering")
>>> question = "What does the customer want?"
>>> answers = reader(question=question, context=text)
>>> answers
{'score': 0.6312923431396484,
 'start': 335,
 'end': 358,
 'answer': 'an exchange of Megatron'}
```

- The task key is `"question-answering"`
- Define your question(s) as a string (or list of strings)
- Pass your question and the text as a context to the pipeline
- The output is a list of dictionaries with answers to questions

Task 5

12 min

Summarization

```
>>> summarizer = pipeline(task="summarization")
>>> summaries = summarizer(
...  text,
...  max_length=45,
...  min_length=45,
...  clean_up_tokenization_spaces=True
... )
>>> summaries
[{"summary_text": "Amazon's Optimus Prime action figure\
wassent to a German online store in Germany. Bumblebee\
asks for an exchange of Megatron for the Optimus Prime\
figure he ordered. He expects to hear from you soon."}]
```

- The task key is `"summarization"`
- Summarize the text using additional arguments
- The output is a list of dictionaries with short summaries of the input text

Splitting and combining strings

```
zen = """The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.

Simple is better than complex.
Complex is better than complicated."""
```

```
>>> parts = zen.split("\n\n")
>>> parts
['The Zen of Python, by Tim Peters',
 'Beautiful is better than ugly.\nExplicit is better than
 'Simple is better than complex.\nComplex is better than c
```

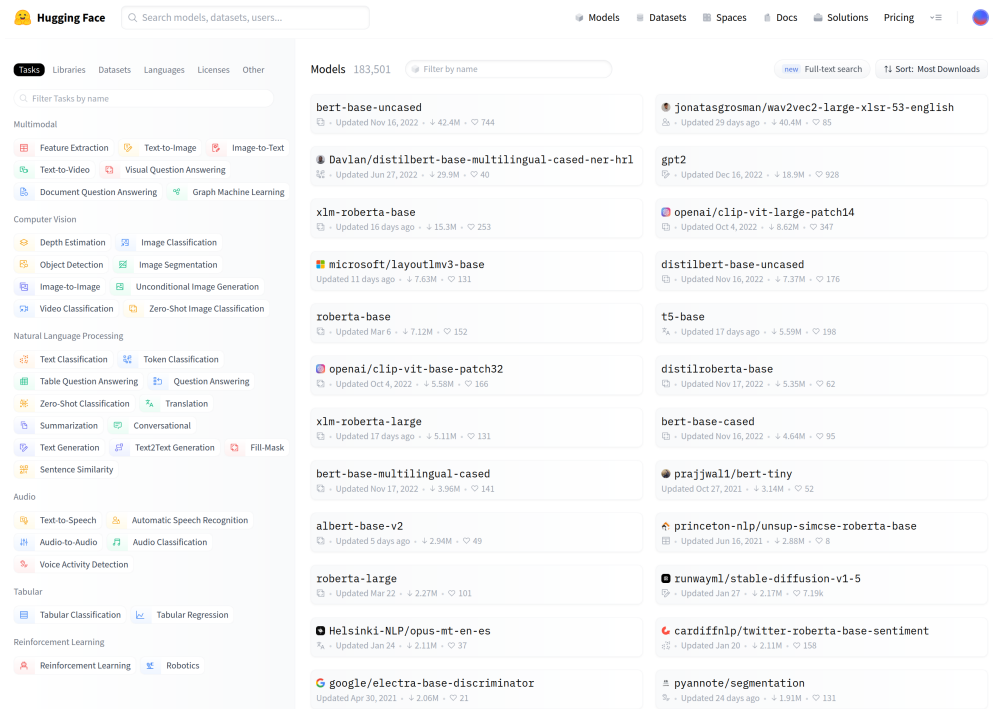
```
>>> print("\n---\n".join(parts))
The Zen of Python, by Tim Peters
---
Beautiful is better than ugly.
Explicit is better than implicit.
---
Simple is better than complex.
Complex is better than complicated.
```

- New lines are represented as `` `\n` `` in python strings
- `` `.split` `` can split a string into a list of strings
- using `` `"\n\n"` `` as delimiter splits by paragraphs
- `` `.join` `` is the inverse of split

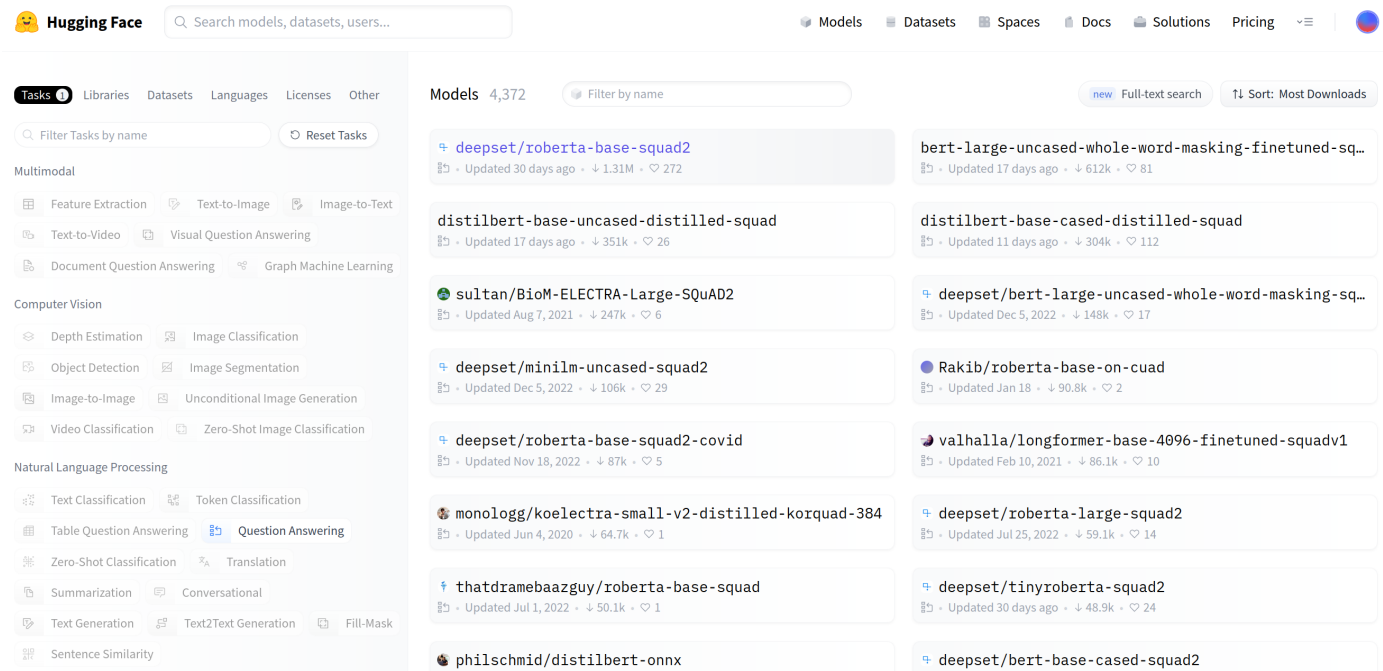
Task 7

(7 min)

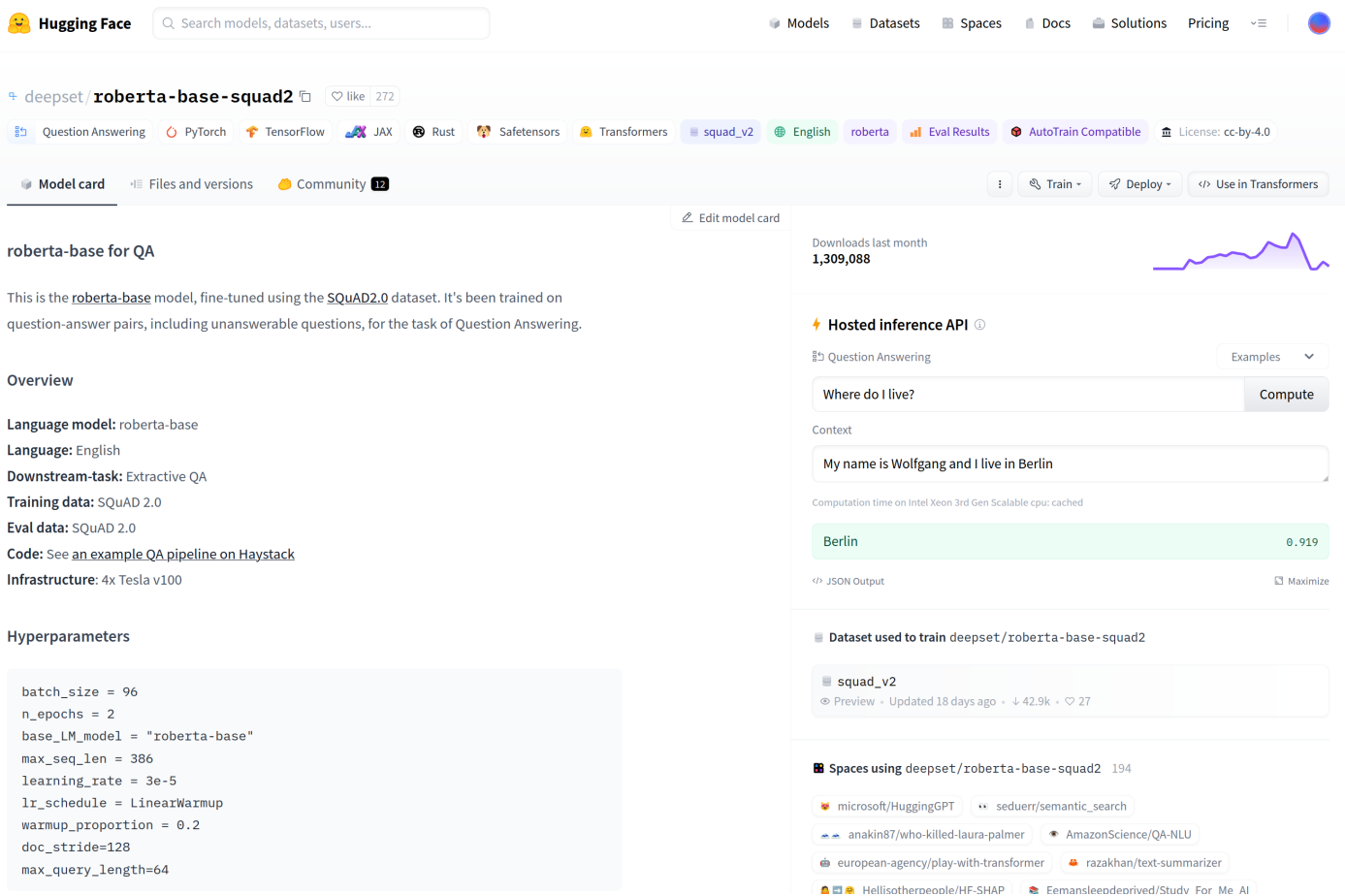
https://huggingface.co/models



Use filters to find models



Check model-card for more information



Translation

```
>>> translator = pipeline(task="translation_en_to_de")
>>> translations = translator(
... text,
... clean_up_tokenization_spaces=True,
... min_lenghth=50
... )
>>> translations
[{"translation_text": "Sehr geehrter Amazon,\
letzte Woche habe ich eine Optimus Prime Action\
Figur aus Ihrem Online-Shop in Deutschland bestellt.\
Leider, als ich das Paket öffnete, entdeckte ich zu\
meinem Entsetzen, dass ich stattdessen eine Action\
Figur von Megatron geschickt worden war! Als\
lebenslanger Feind der Decepticons, Ich hoffe,\
Sie können mein Dilemma verstehen. Um das Problem\
zu lösen, Ich fordere einen Austausch von Megatron\
für die Optimus Prime Figur habe ich bestellt."}]
```

- The task key is `` `"translation_xx_to_yy"` ``
- Translate your text using additional arguments
- The output is a list of dictionaries with the translated text

Task 8

10 min