# Hugging Face Models

Venkata Reddy AI Classes

https://www.youtube.com/@VenkataReddyAIClasses/playlists

# Contents

- Open source LLMs
- Model Hub on Hugging face
- Model from Hugging face
- pipeline() function
- Document Classification model
- Q&A Model
- NER Model
- Saving the Model on HuggingFace hub
- Final app – Bank Customers Complaints categorization App

# GenAI Pre-requisites

MIS Reports

Basic Reporting

Basic Predictions

Predictive Modeling

Advanced Predictions

Machine Learning

Image and text classification

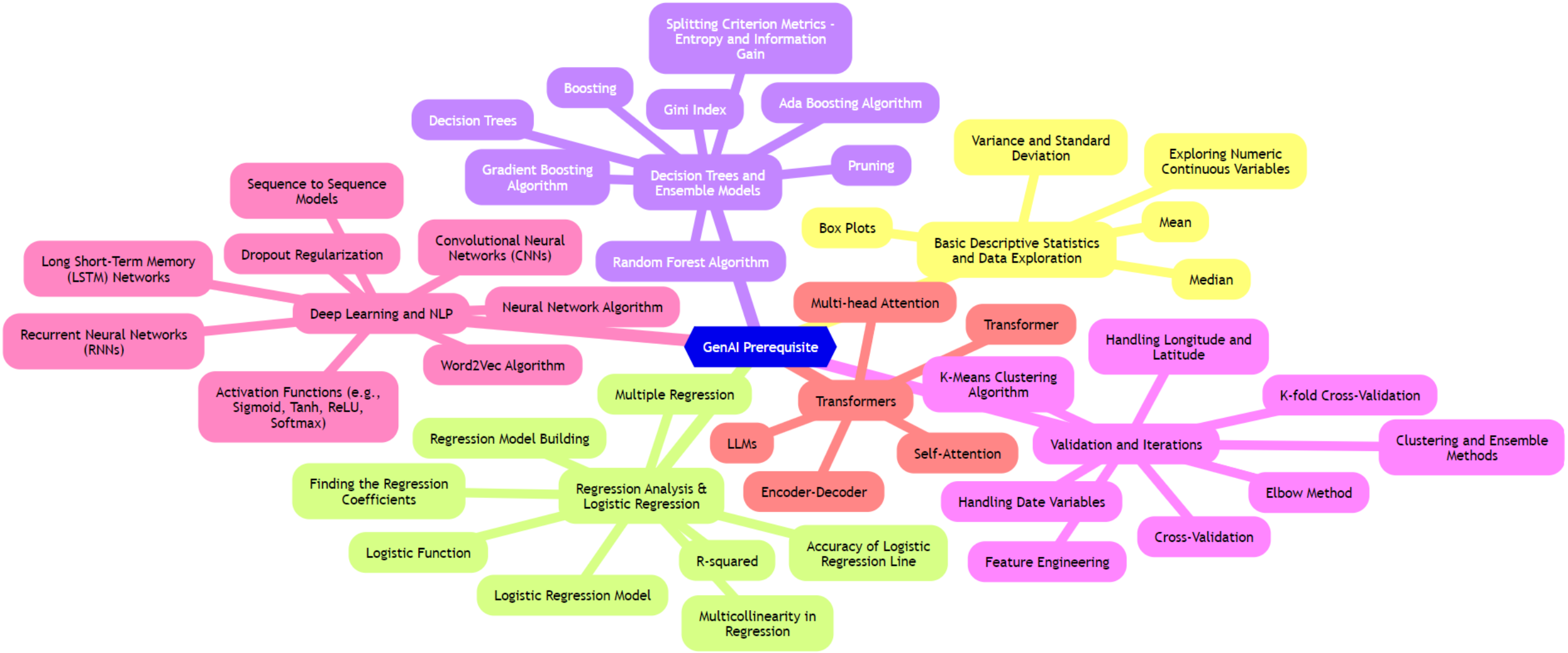
Deep Learning

Image and Text Generation

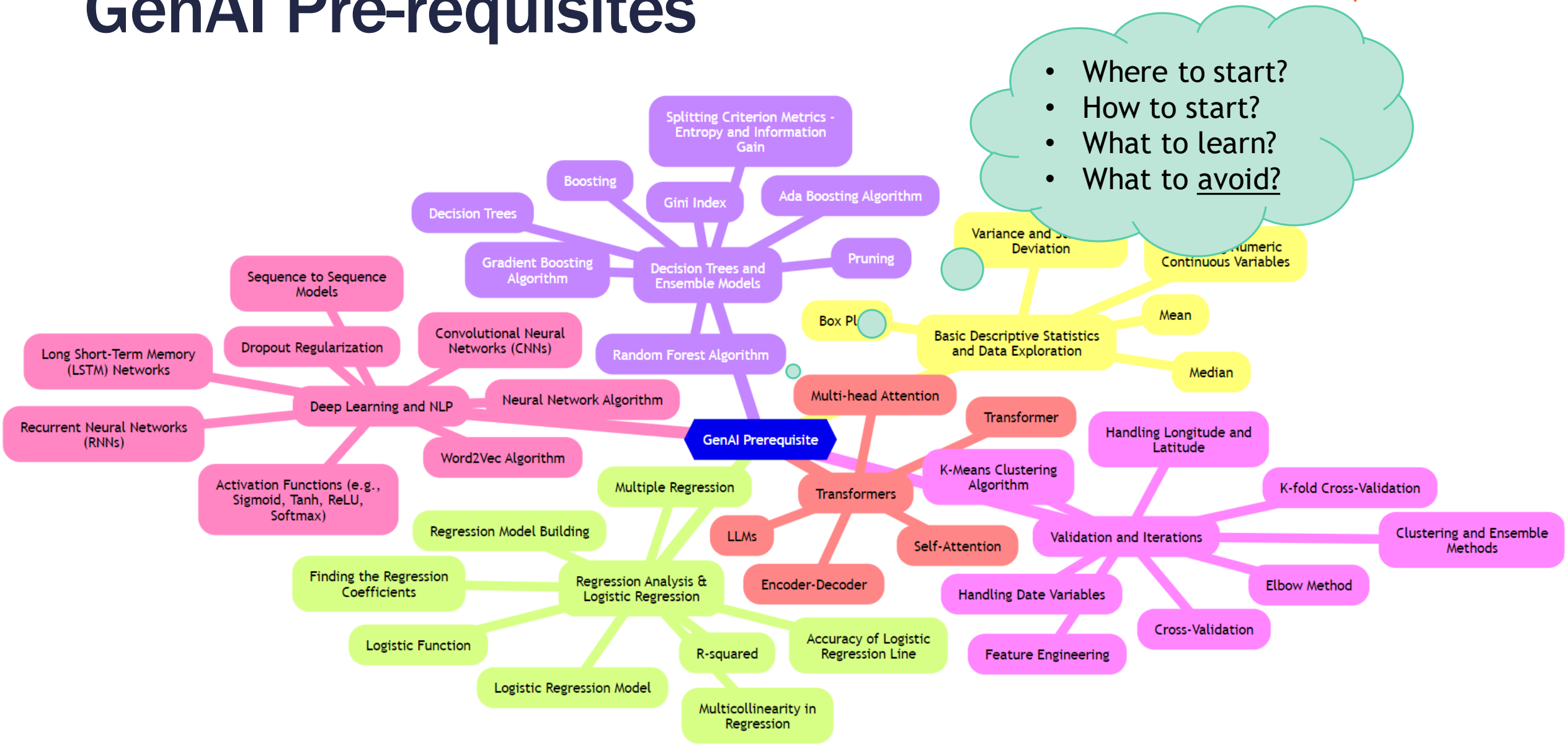
GenAI

There is a significant volume of information here. Require at least six months to explore it

# GenAI Pre-requisites



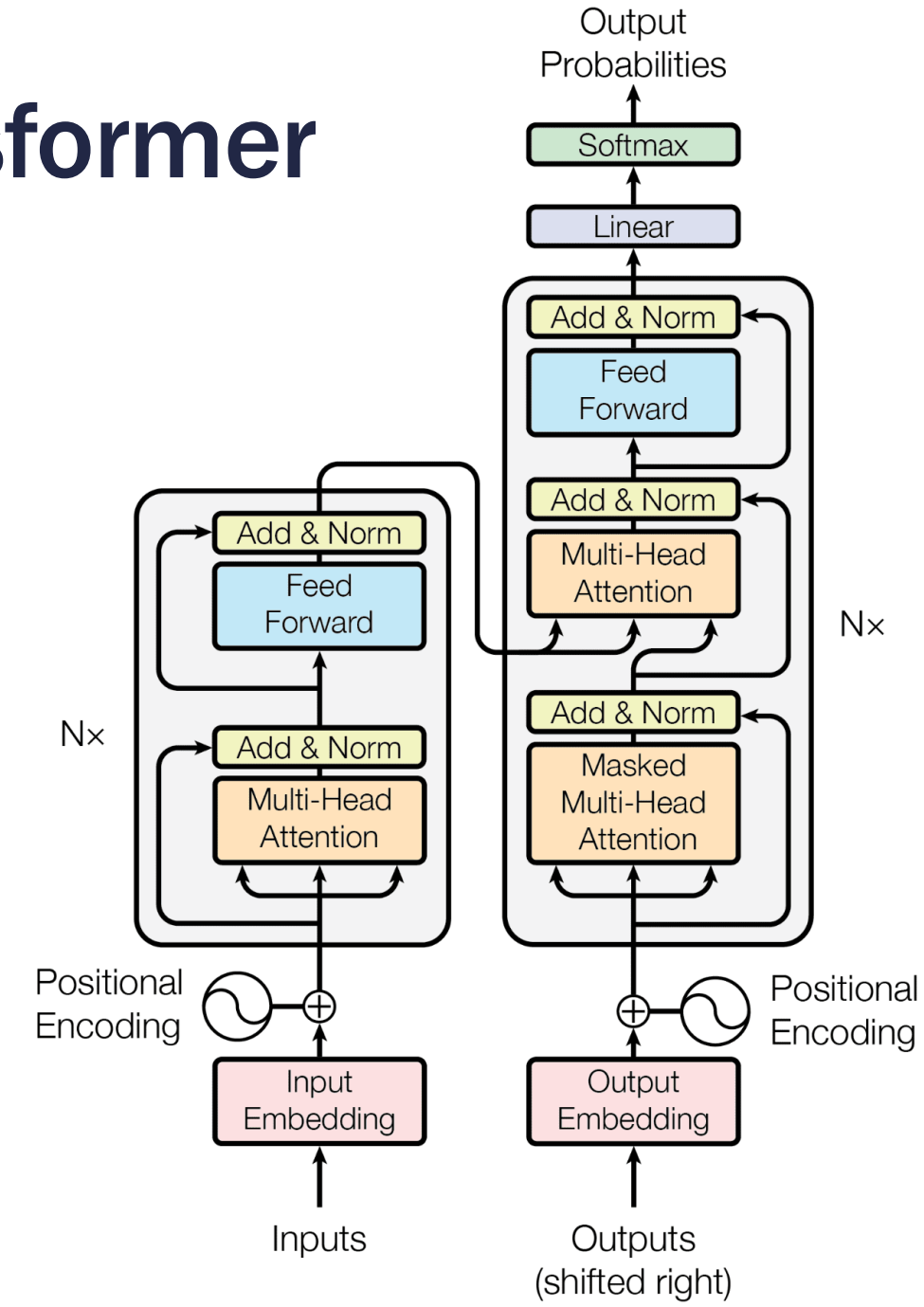
# GenAI Pre-requisites



# GenAI Pre-requisites

# Almost every LLM is a Transformer

- Transformer is an advanced Deep Neural Network model
- Transformers revolutionize NLP, forming the basis for modern LLMs.
- Designed to process and understand sequential data like text.
- Capture relationships between words regardless of their position.
- Attention mechanism allows models to focus on all input parts simultaneously.



# What are the opensource LLMs?

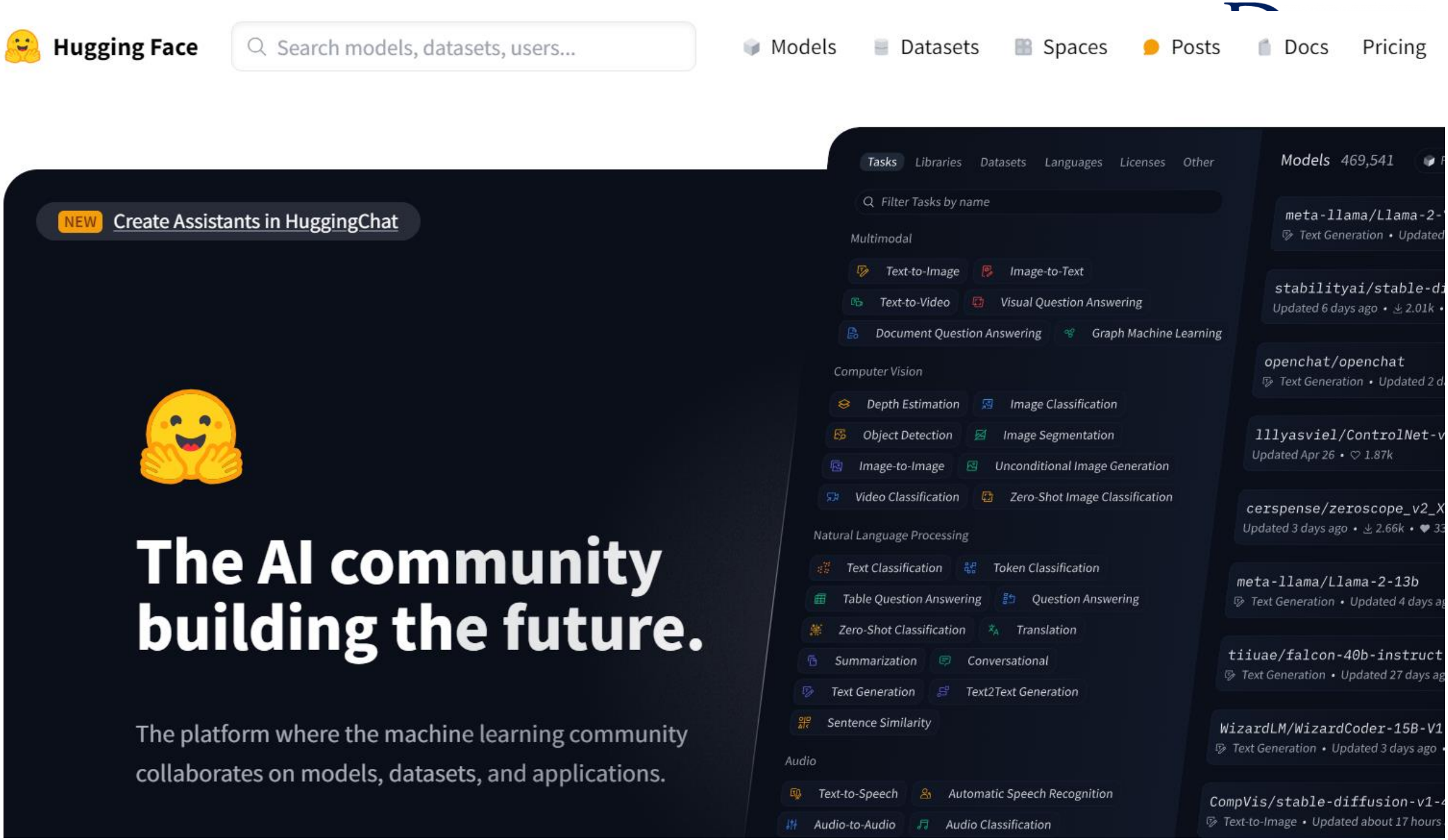
- What are the opensource LLMs?
- How to access them using APIs?
- Where to find them?
- Where can we find the documentation ?
- Is there any repository where I can find all the LLMs ? - Yes
- **Hugging Face**

# Hugging Face

- If you are working with language models, you have to be comfortable with Hugging Face hub
- Hugging face is the repository for several pre-trained ML, NLP, Vision based models and datasets.
- You can access various models and datasets.
- You can also upload your own model into hugging face platform

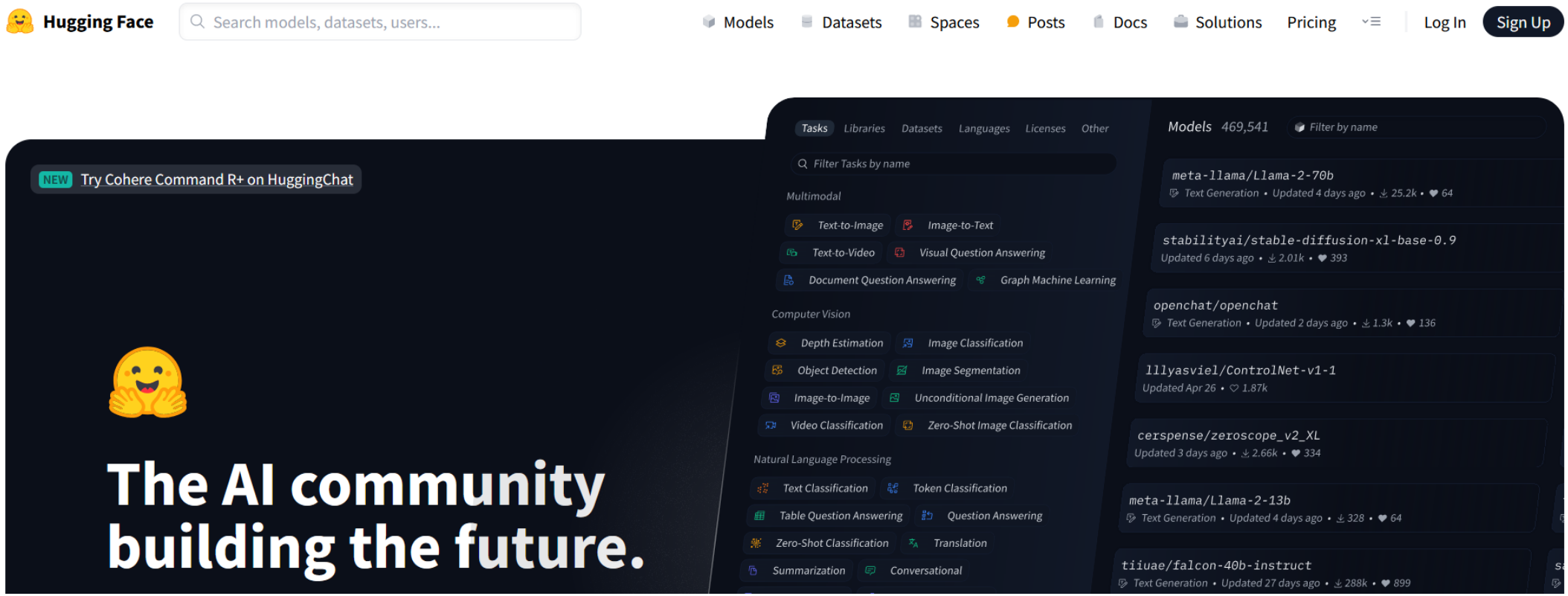
# Repository of all ML and AI models

- Designed for sharing, collaborating, and managing machine learning models, datasets, etc.
- Trying to democratize machine learning, especially in NLP and computer vision. Make the models available for everyone
- Contains good documentation and example codes
- **You must learn** how to use Hugging Face models before jumping on to LLMs

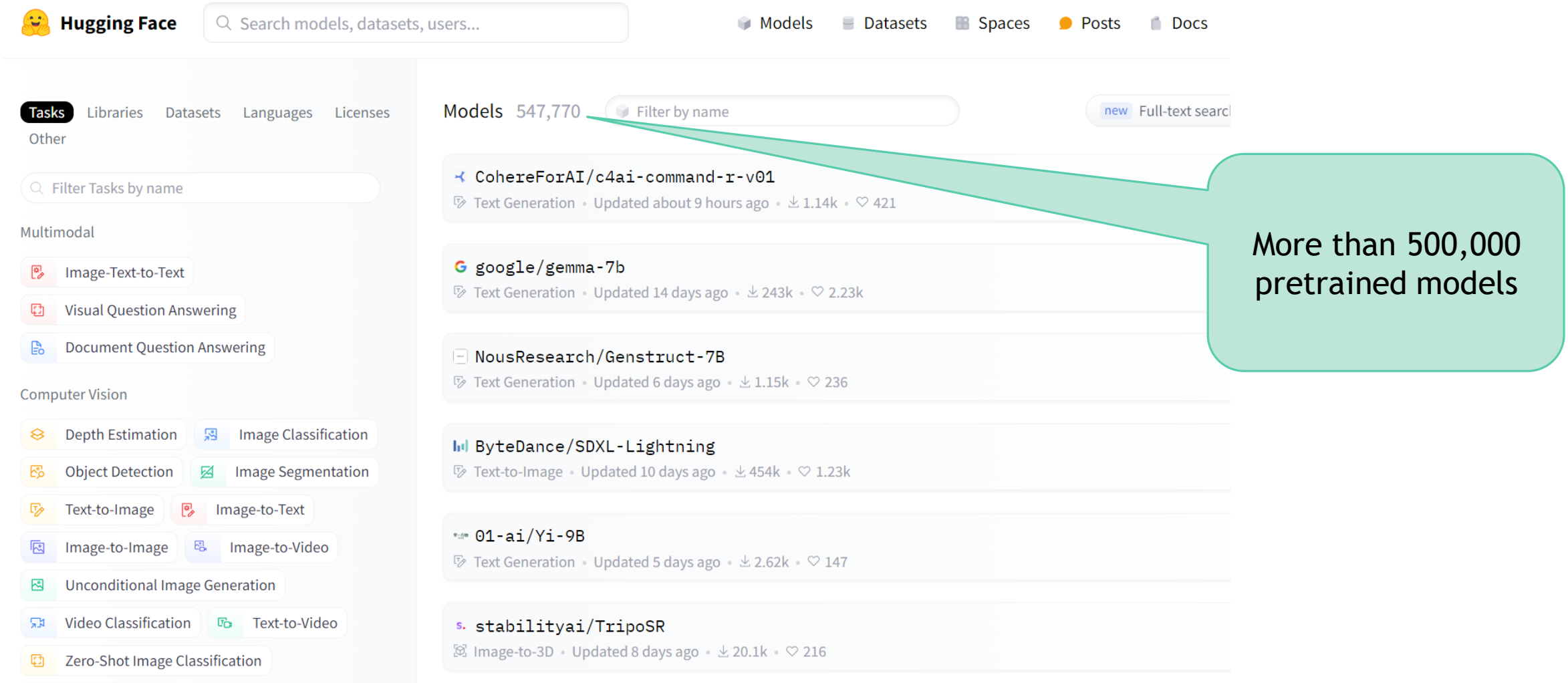

Hugging Face
Search models, datasets, users...
Models
Datasets
Spaces
Posts
Docs
Pricing
NEW
Create Assistants in HuggingChat
The AI community building the future.
The platform where the machine learning community collaborates on models, datasets, and applications.
Tasks
Libraries
Datasets
Languages
Licenses
Other
Filter Tasks by name
Multimodal
Text-to-Image
Image-to-Text
Text-to-Video
Visual Question Answering
Document Question Answering
Graph Machine Learning
Computer Vision
Depth Estimation
Image Classification
Object Detection
Image Segmentation
Image-to-Image
Unconditional Image Generation
Video Classification
Zero-Shot Image Classification
Natural Language Processing
Text Classification
Token Classification
Table Question Answering
Question Answering
Zero-Shot Classification
Translation
Summarization
Conversational
Text Generation
Text2Text Generation
Sentence Similarity
Audio
Text-to-Speech
Automatic Speech Recognition
Audio-to-Audio
Audio Classification
Models 469,541
meta-llama/Llama-2-
Text Generation • Updated
stabilityai/stable-di
Updated 6 days ago • 2.01k
openchat/openchat
Text Generation • Updated 2 d
lllyasviel/ControlNet-v
Updated Apr 26 • 1.87k
cerspense/zeroscope_v2_X
Updated 3 days ago • 2.66k
meta-llama/Llama-2-13b
Text Generation • Updated 4 days ag
tiiuae/falcon-40b-instruct
Text Generation • Updated 27 days ag
WizardLM/WizardCoder-15B-V1
Text Generation • Updated 3 days ago
CompVis/stable-diffusion-v1-
Text-to-Image • Updated about 17 hours

# Sign-up

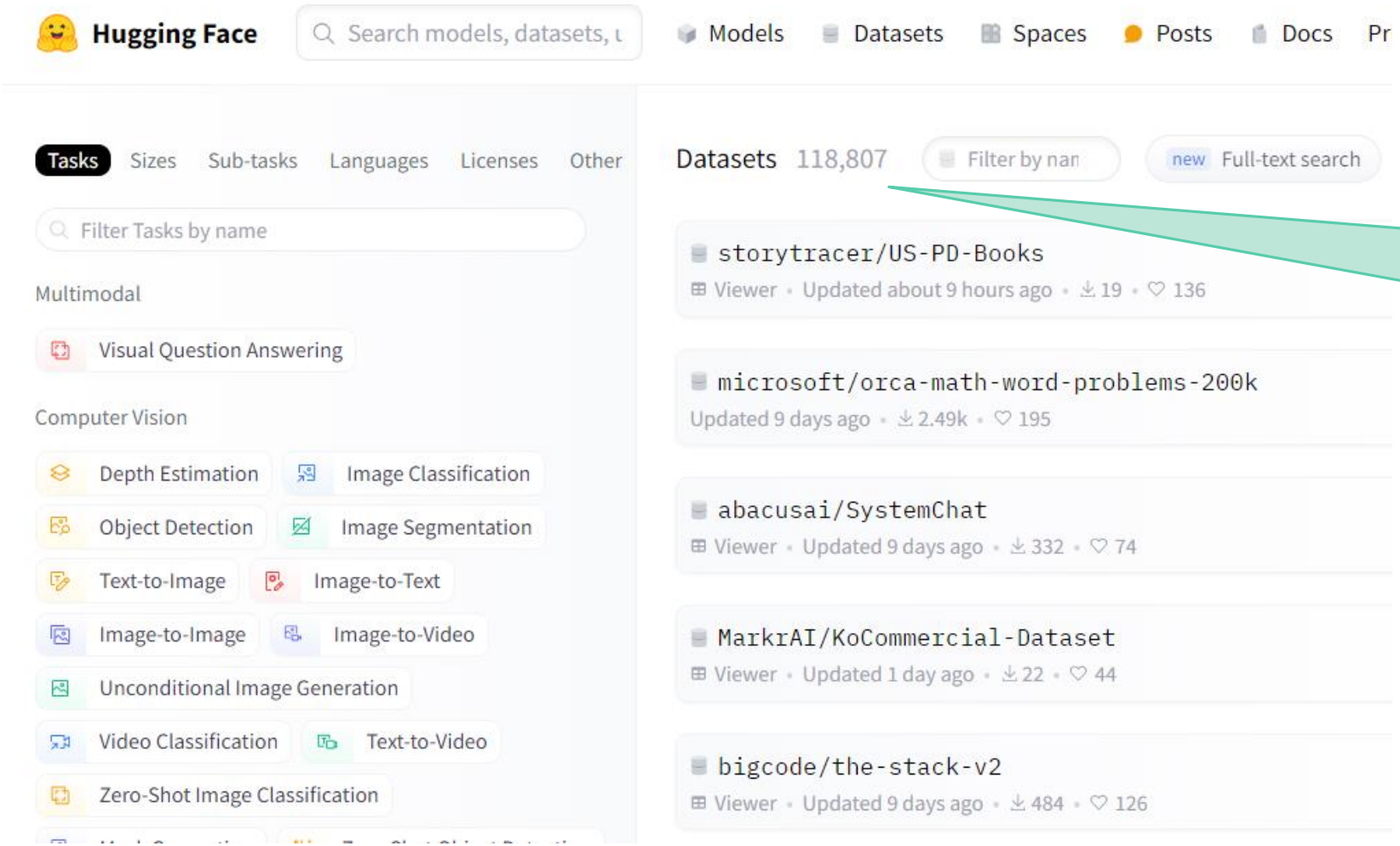
Signup and create a login-id



# Model Hub contains a huge collection of pre-trained models

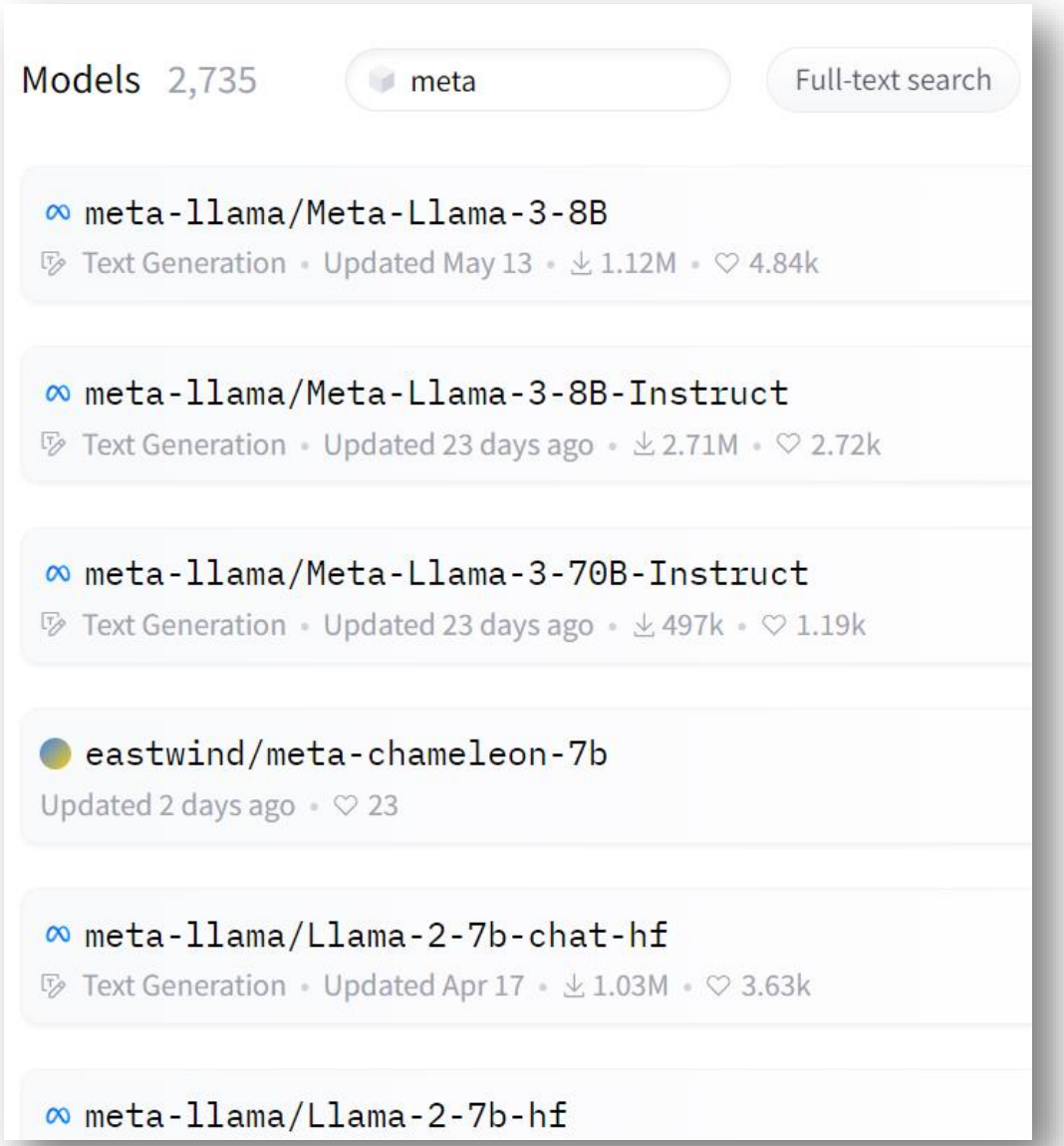


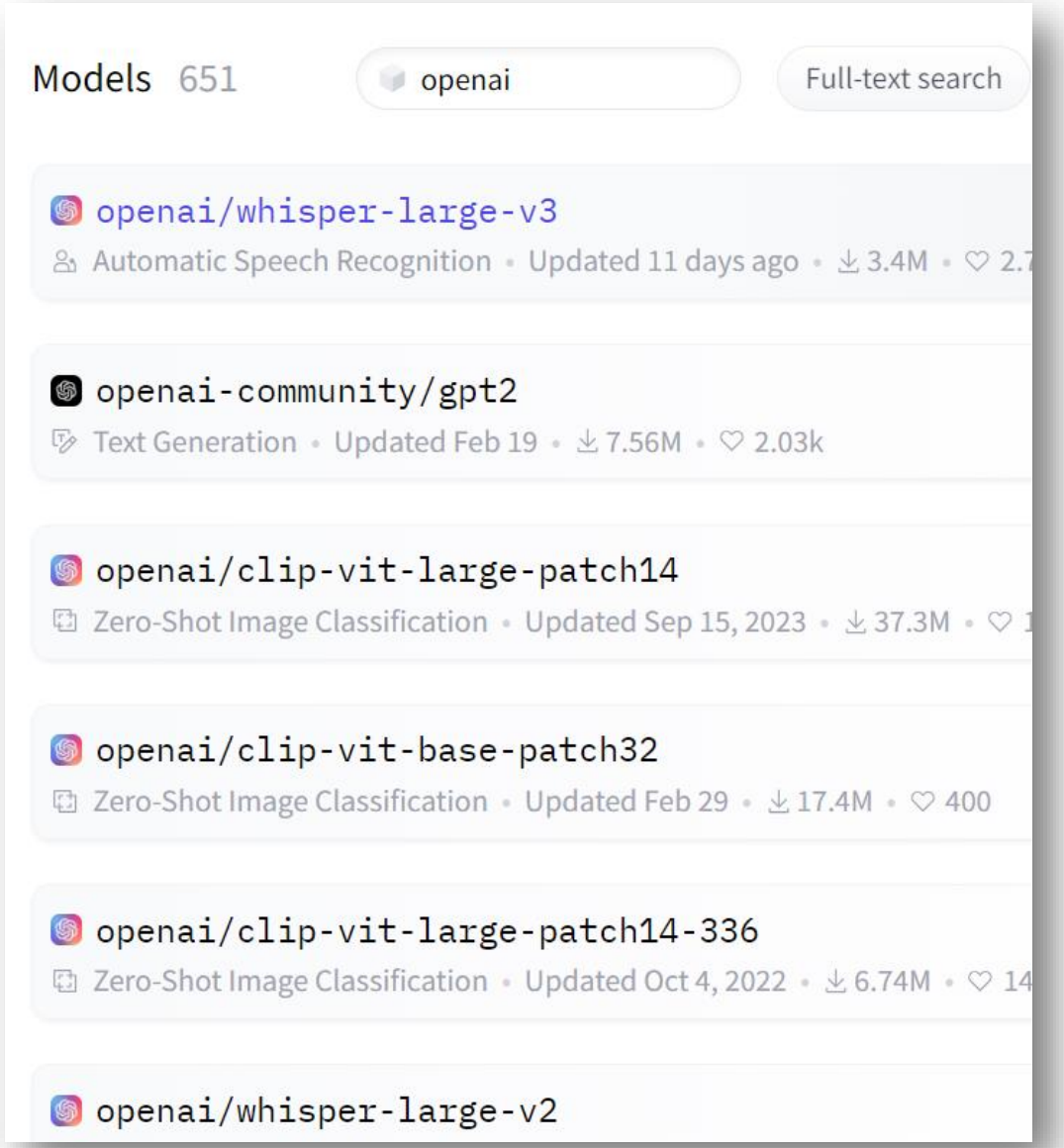
# A huge collection of datasets also



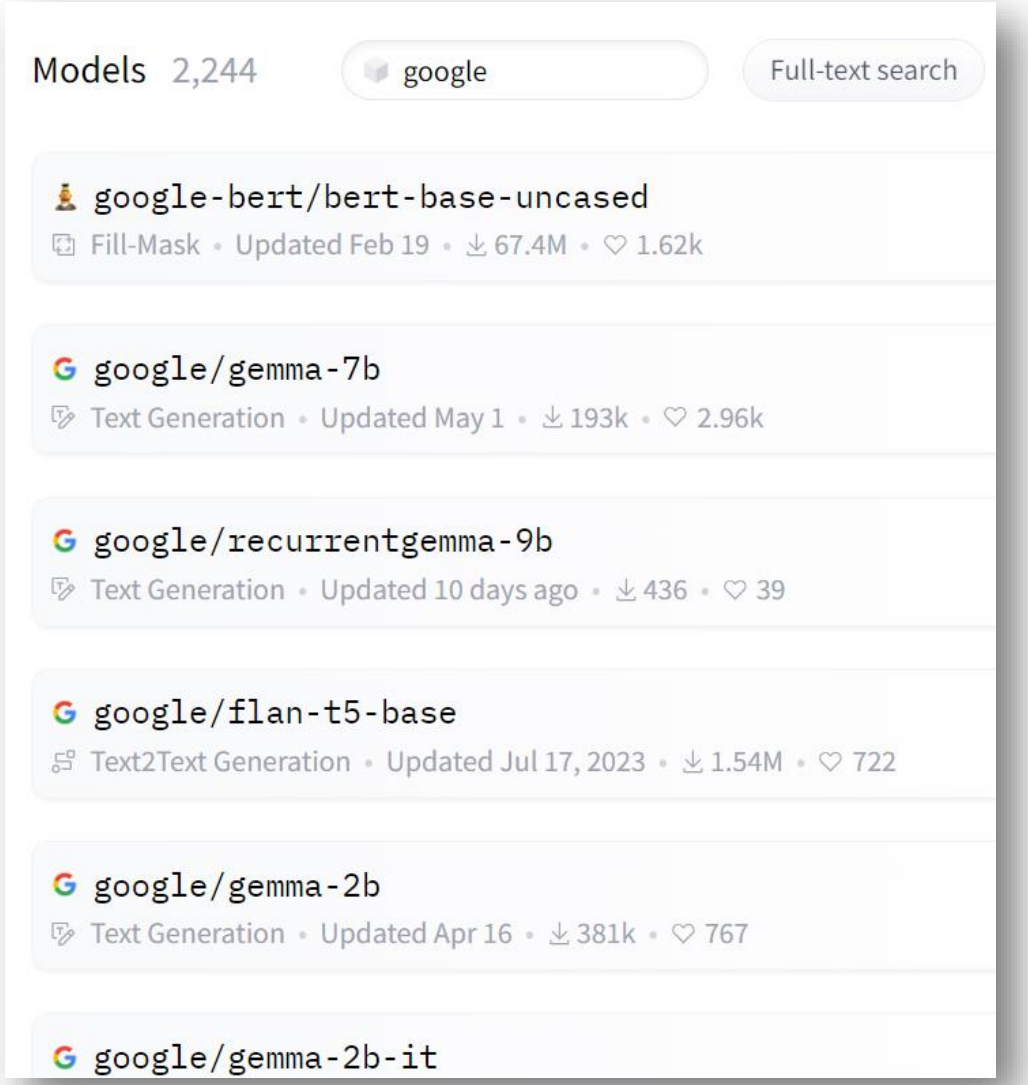

More than 100,000 datasets

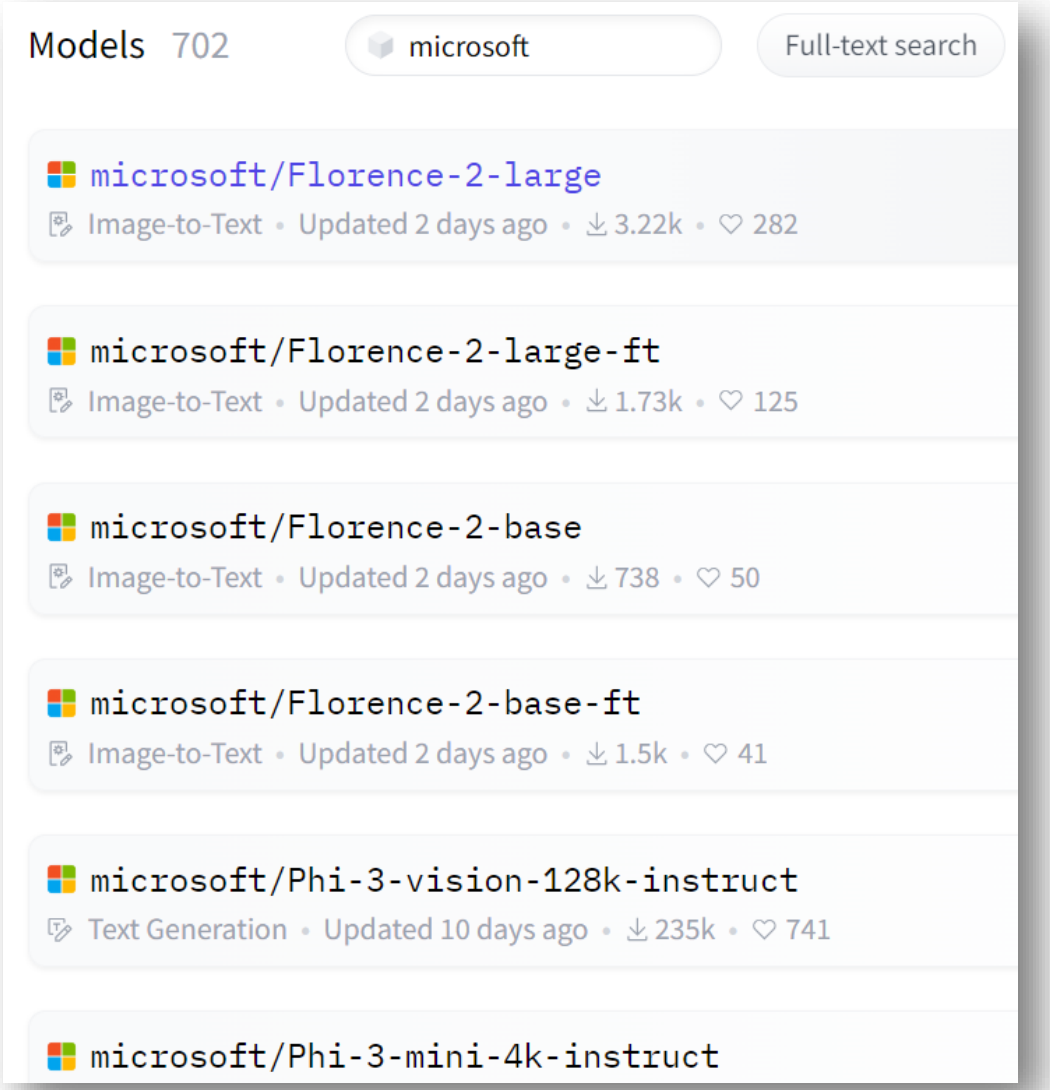
# Model Hub on Hugging Face



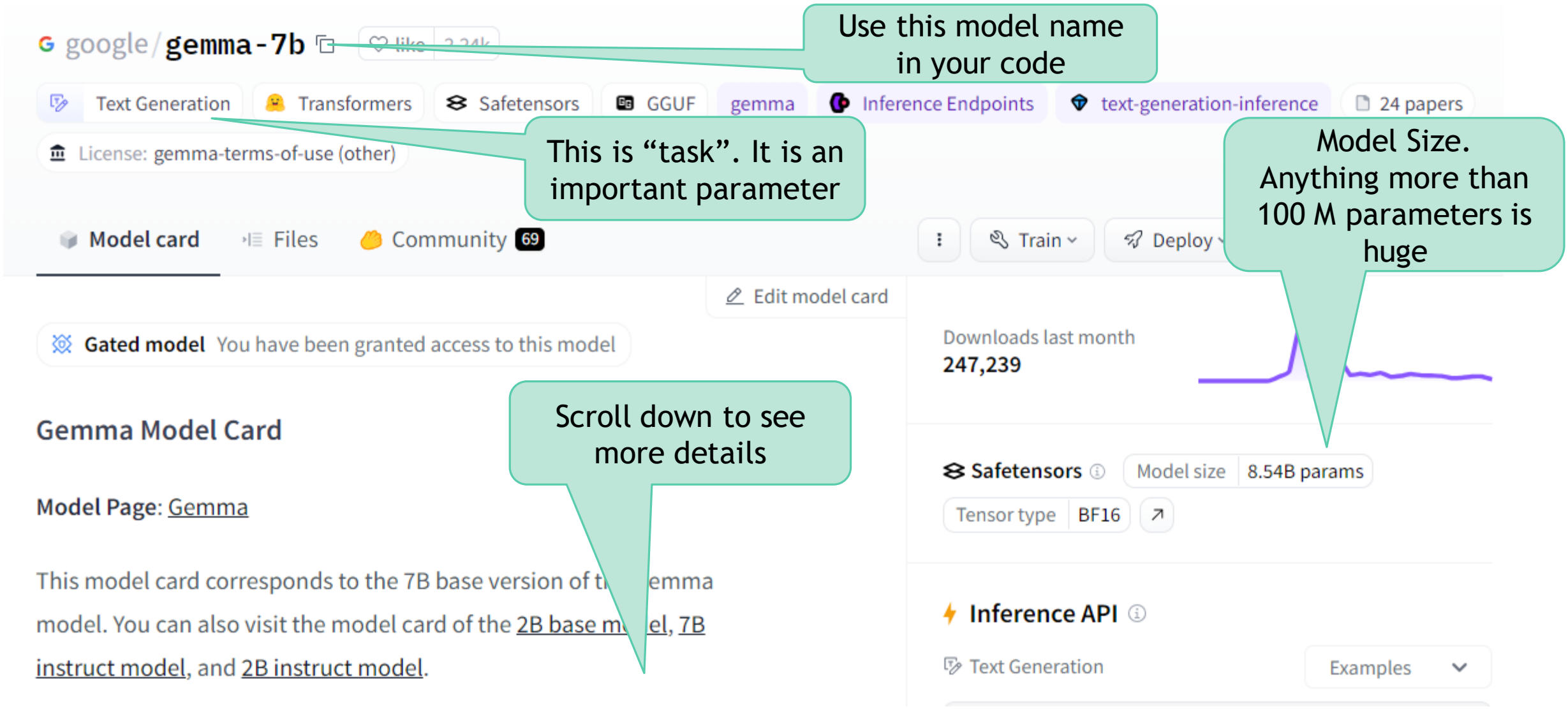



# Model Hub on Hugging Face

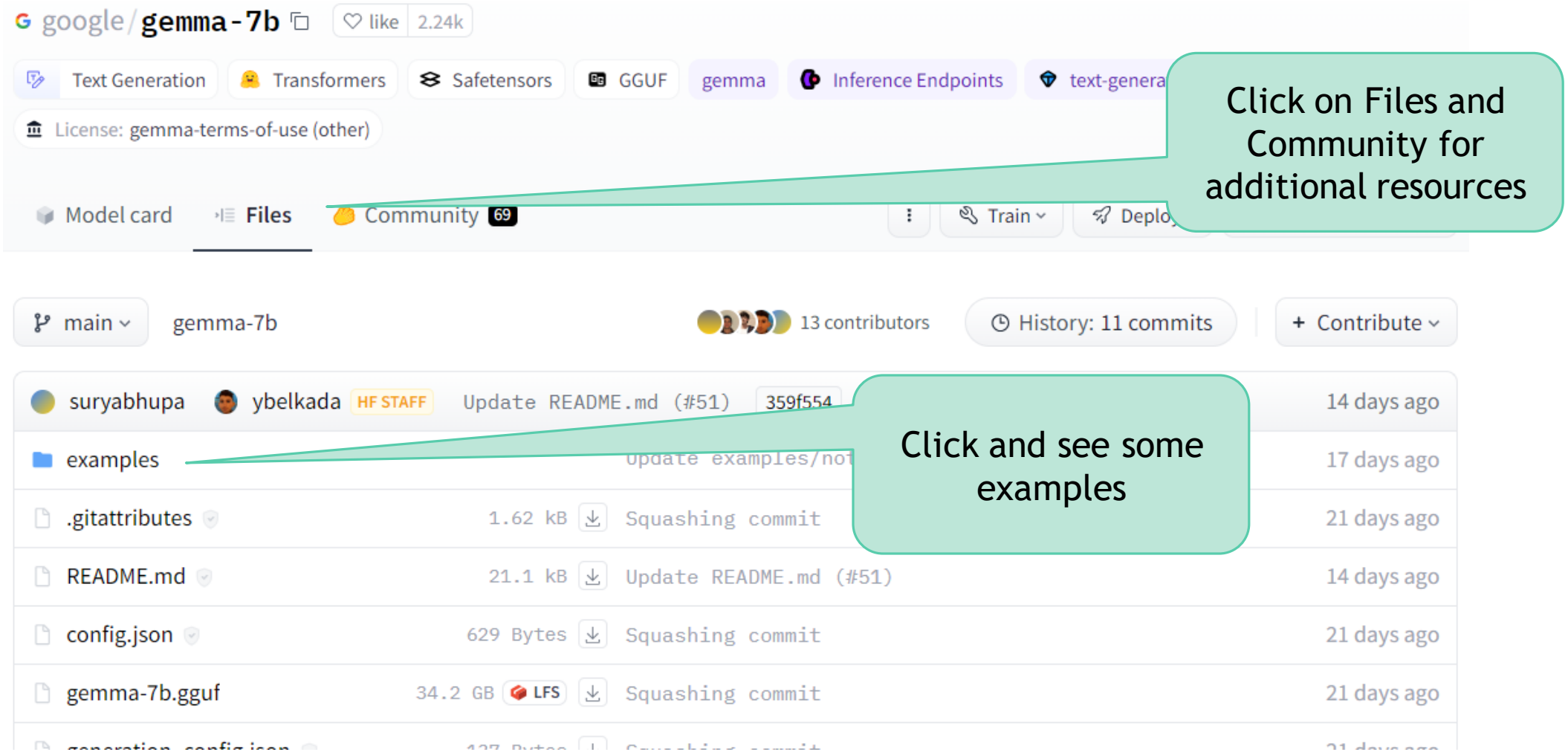


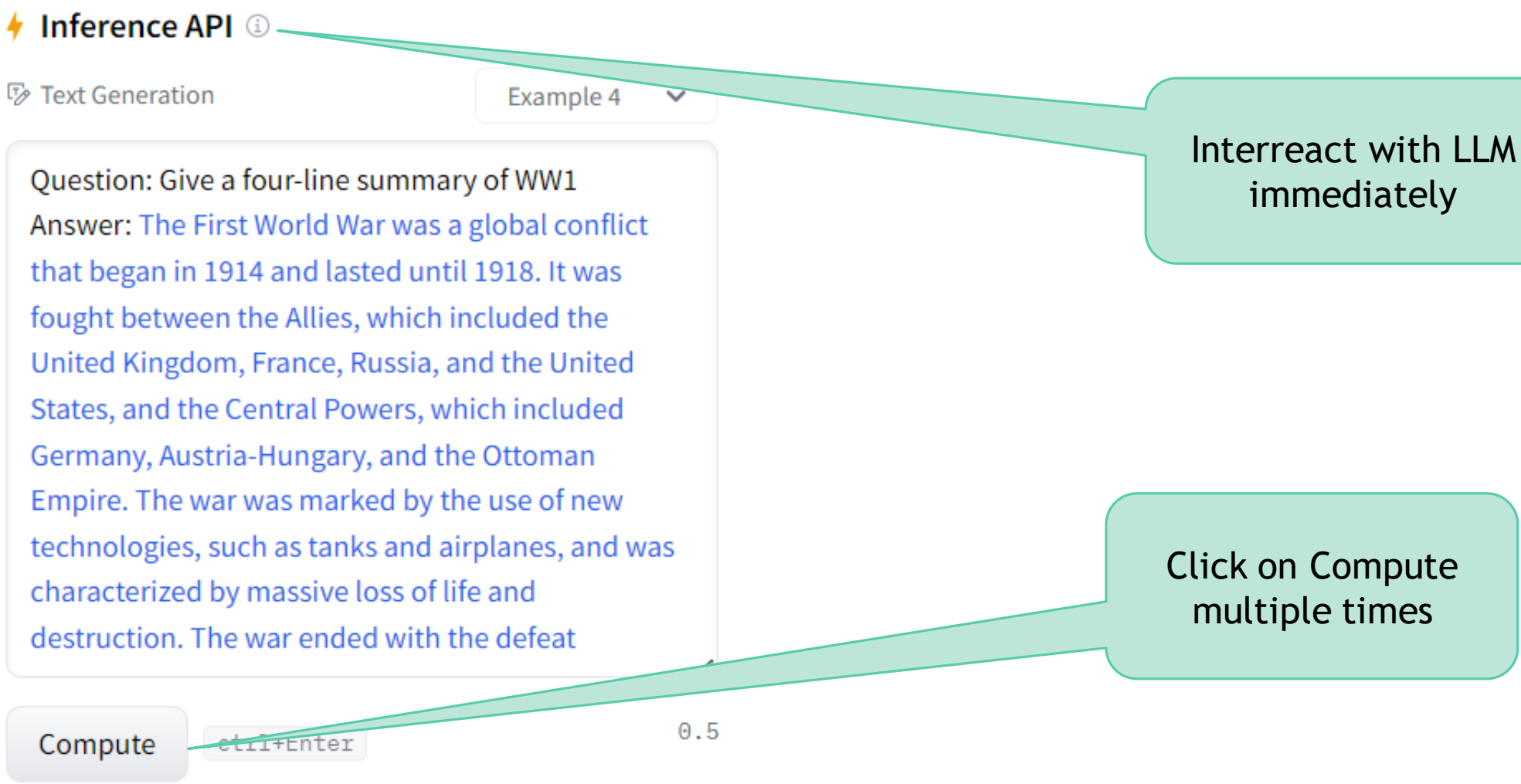
# Sample Model from Hugging face



# Model from Hugging face



# Model from Hugging face



# Model from Hugging face

Running the model on a CPU

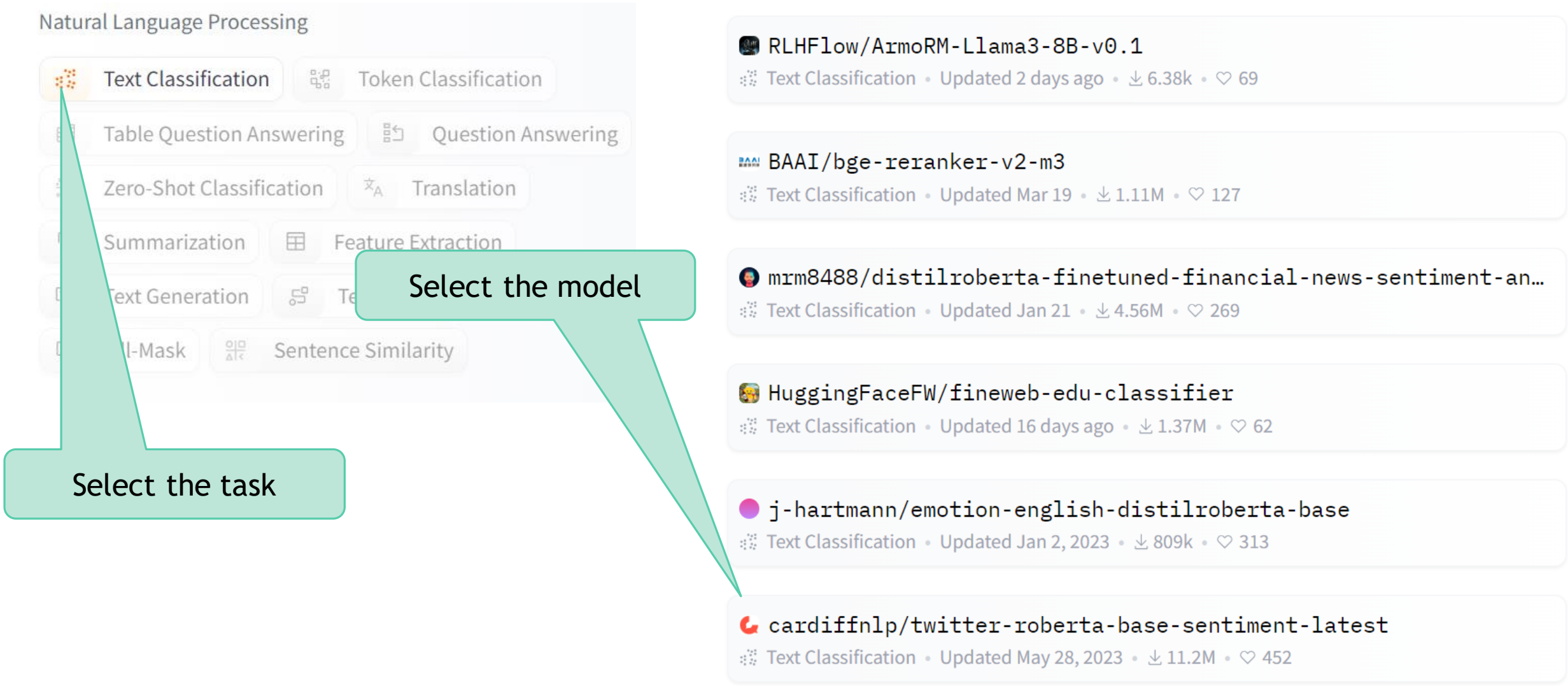
```
from transformers import AutoTokenizer, AutoModelForCa

tokenizer = AutoTokenizer.from_pretrained("google/gemm
model = AutoModelForCausalLM.from_pretrained("google/g

input_text = "Write me a poem about Machine Learning."
input_ids = tokenizer(input_text, return_tensors="pt")

outputs = model.generate(**input_ids)
print(tokenizer.decode(outputs[0]))
```

Running the model on a single / multi GPU

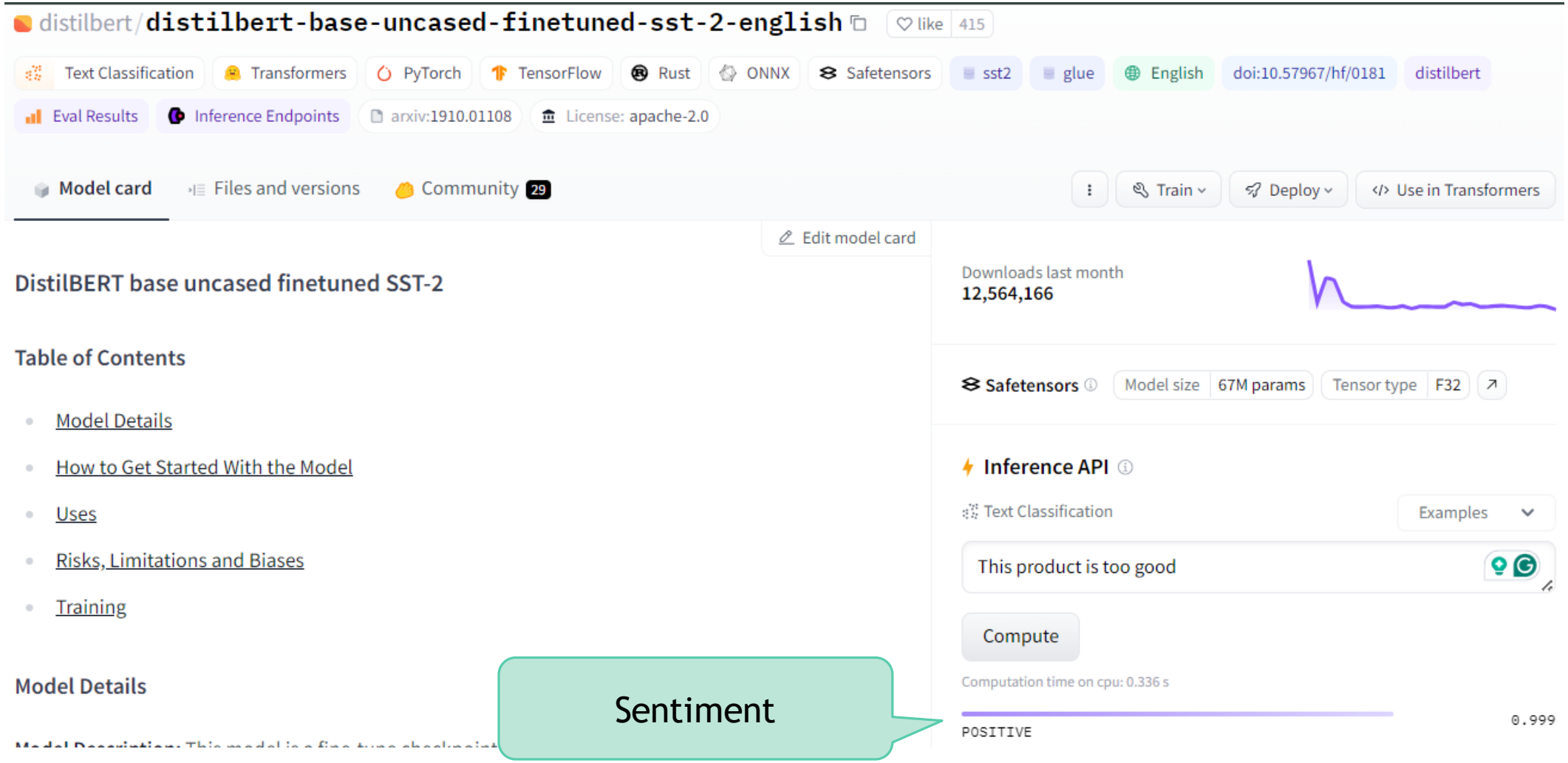
```
# pip install accelerate
from transformers import AutoTokenizer, AutoModelForCa

tokenizer = AutoTokenizer.from_pretrained("google/gemm
model = AutoModelForCausalLM.from_pretrained("google/g

input_text = "Write me a poem about Machine Learning."
input_ids = tokenizer(input_text, return_tensors="pt")

outputs = model.generate(**input_ids)
print(tokenizer.decode(outputs[0]))
```

Code samples provided as part of documentation

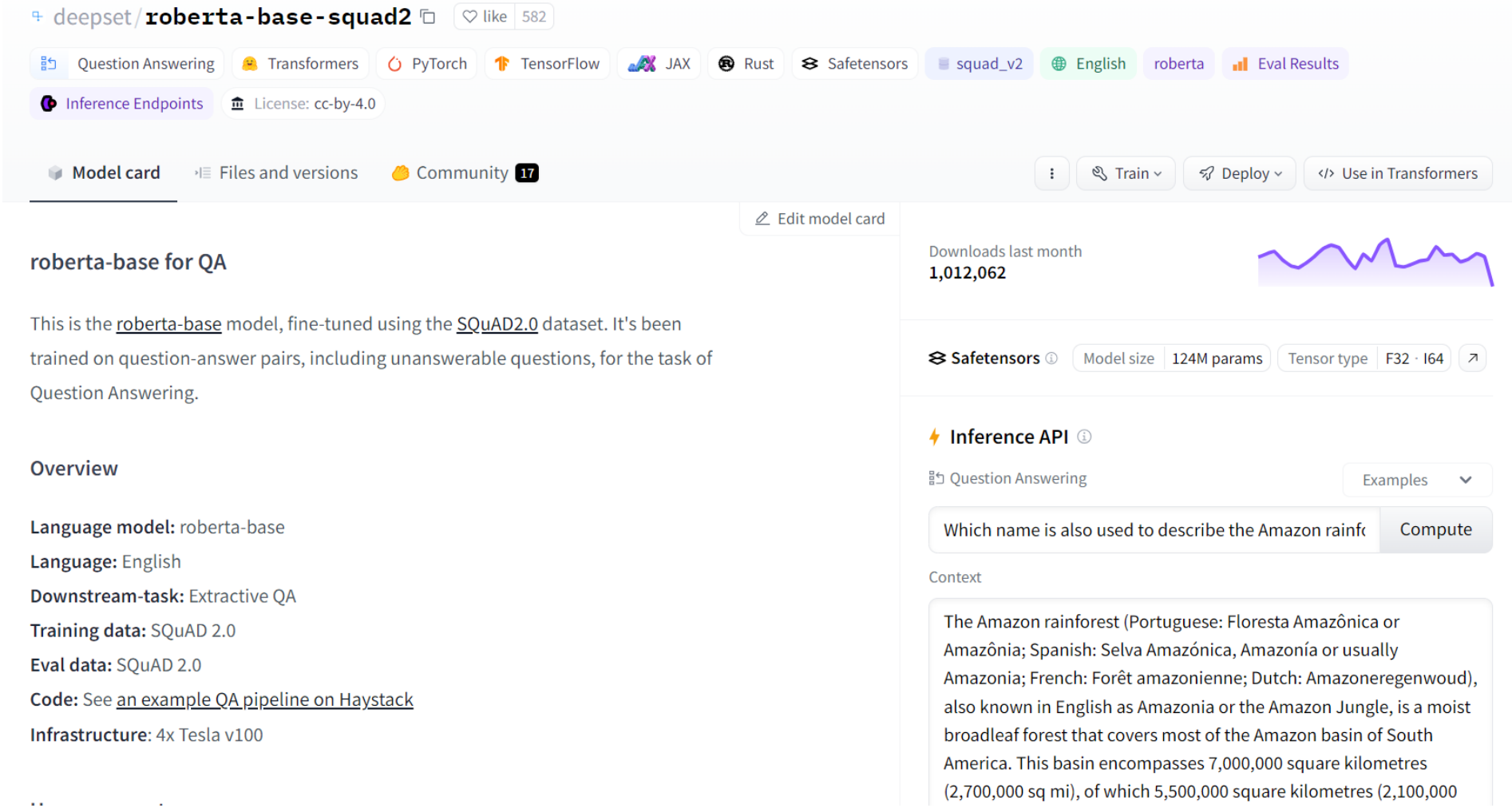
# Text Classification - sentiment analysis



# distilbert model for sentiment analysis



# deepset/roberta-base-squad2



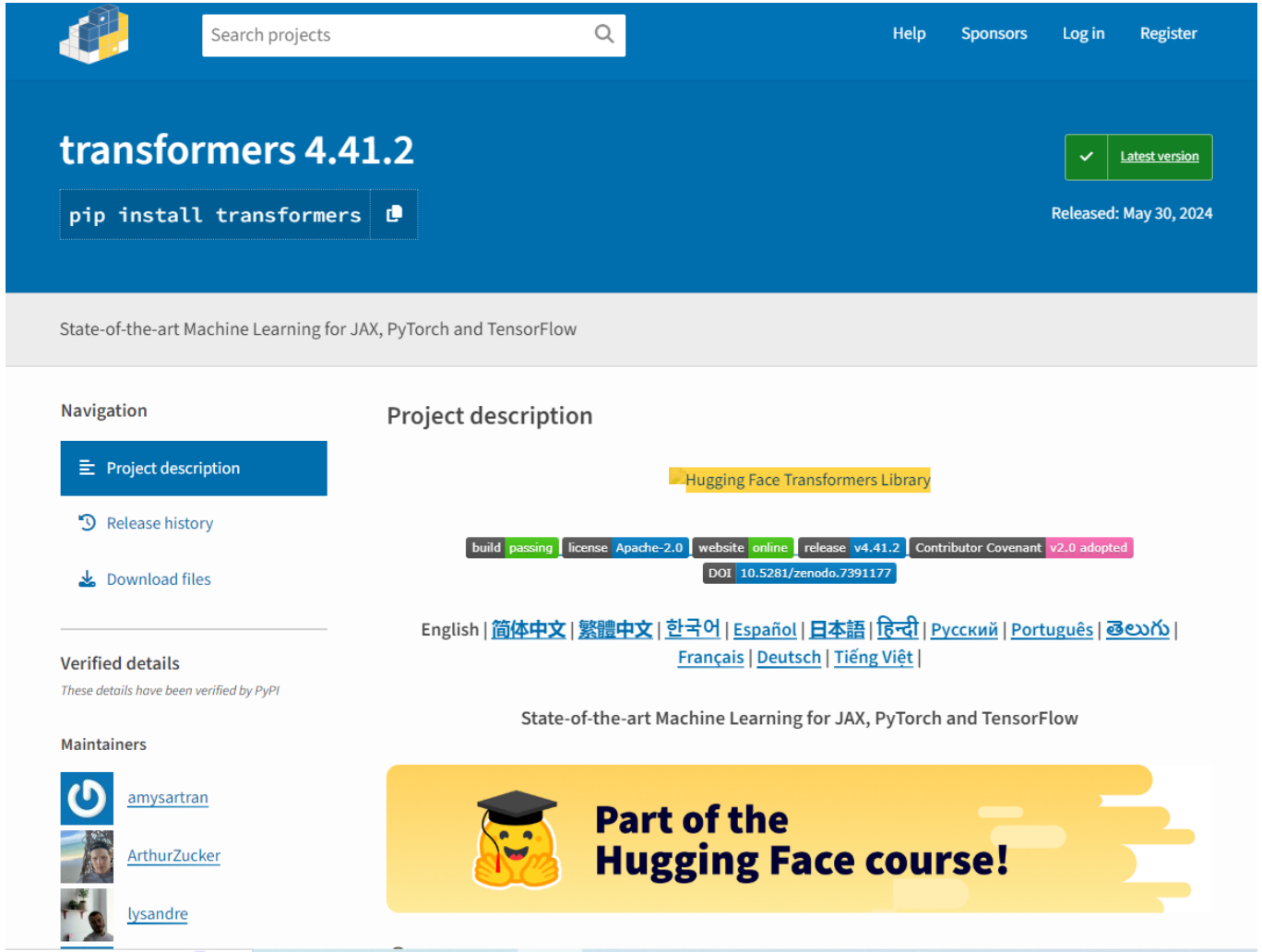
# Hugging face `pipeline()`
## To get started quickly

```
from transformers import AutoTokenizer, AutoModelForCa

tokenizer = AutoTokenizer.from_pretrained("google/gemm
model = AutoModelForCausalLM.from_pretrained("google/g

input_text = "Write me a poem about Machine Learning."
input_ids = tokenizer(input_text, return_tensors="pt")

outputs = model.generate(**input_ids)
print(tokenizer.decode(outputs[0]))
```

- This is the original code, but there is an easy way to call this model
- We will use pipeline() function

# About the transformers package

- Transformers is a community of projects and the Hugging Face Hub.
- It supports developers, researchers, students, professors, and engineers.
- Enables users to build their dream projects with ease.
- Provides thousands of pre-trained models for various tasks.
- Supports different modalities like text, vision, and audio.



# About the `pipeline()` function

- The pipeline package in the transformers library simplifies the process of using pre-trained NLP models
- It helps us by abstracting away complex steps like tokenization, model loading, and post-processing.
- Almost similar code can be used for multiple pre-trained models.
- The pipeline() function takes care of the necessary steps, including tokenizing input text, passing it through the model, and returning the results in a user-friendly format.

```
from transformers import pipeline
```

# Sentiment Analysis Model

```
senti_model = pipeline(task="sentiment-analysis")
```

- There are multiple models for sentiment analysis
- We have not specified any model name
- The default model will be considered

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-
english and revision af0f99b (https://huggingface.co/distilbert/distilbert-base-uncased-
finetuned-sst-2-english).
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 secret in your Google Colab and restart
your session.
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(
```

- Hugging face provides an API key. We have not mentioned it here.
- It still works for some models

# Sentiment Analysis Model - Result

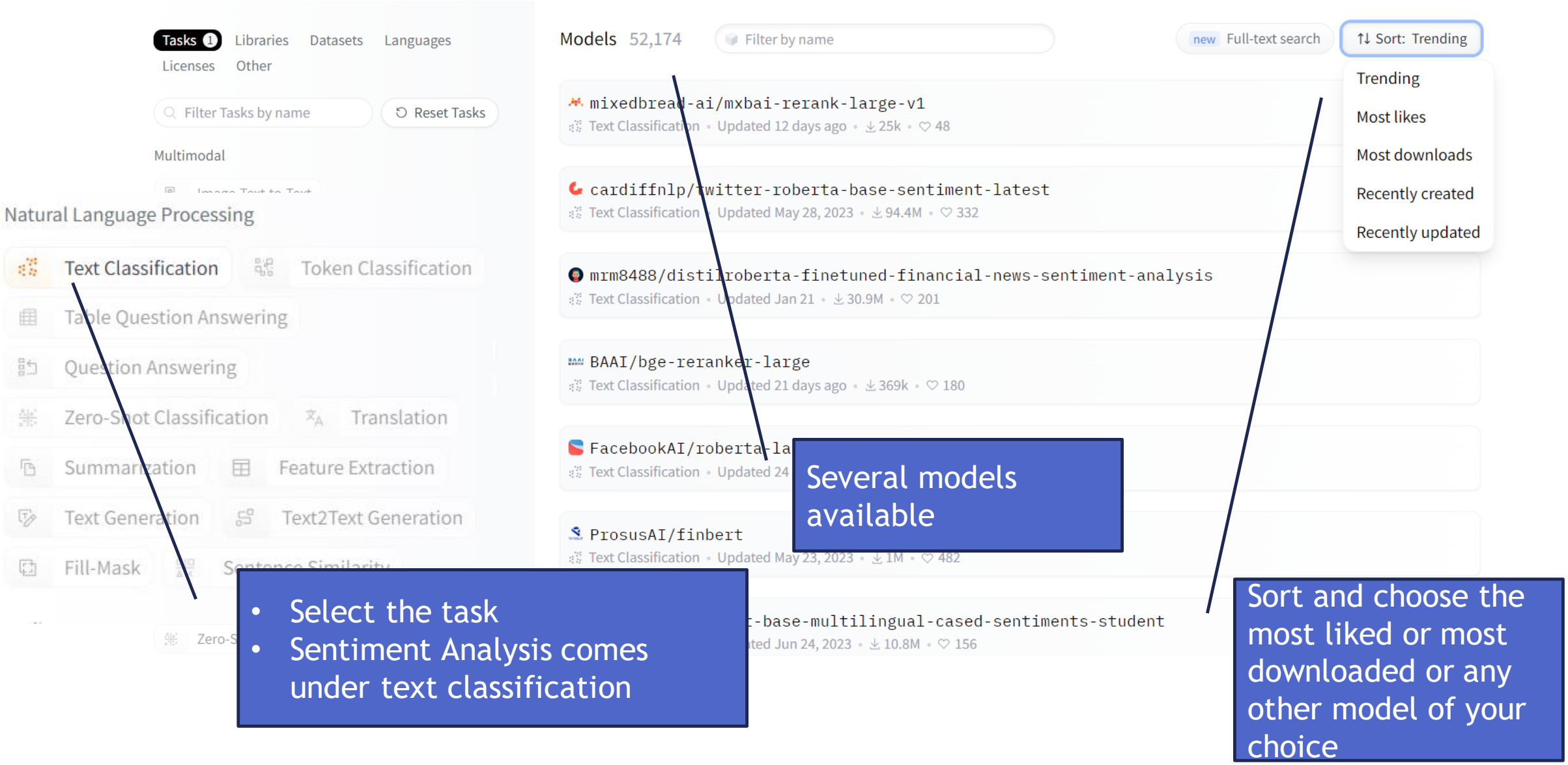
```
senti_model("This movie is damn good. I loved it")
```

```
[{'label': 'POSITIVE', 'score': 0.9998770952224731}]
```

```
senti_model("This is a bad phone. The screen and battery are of poor quality.")
```

```
[{'label': 'NEGATIVE', 'score': 0.9998168349266052}]
```

# Choose your model

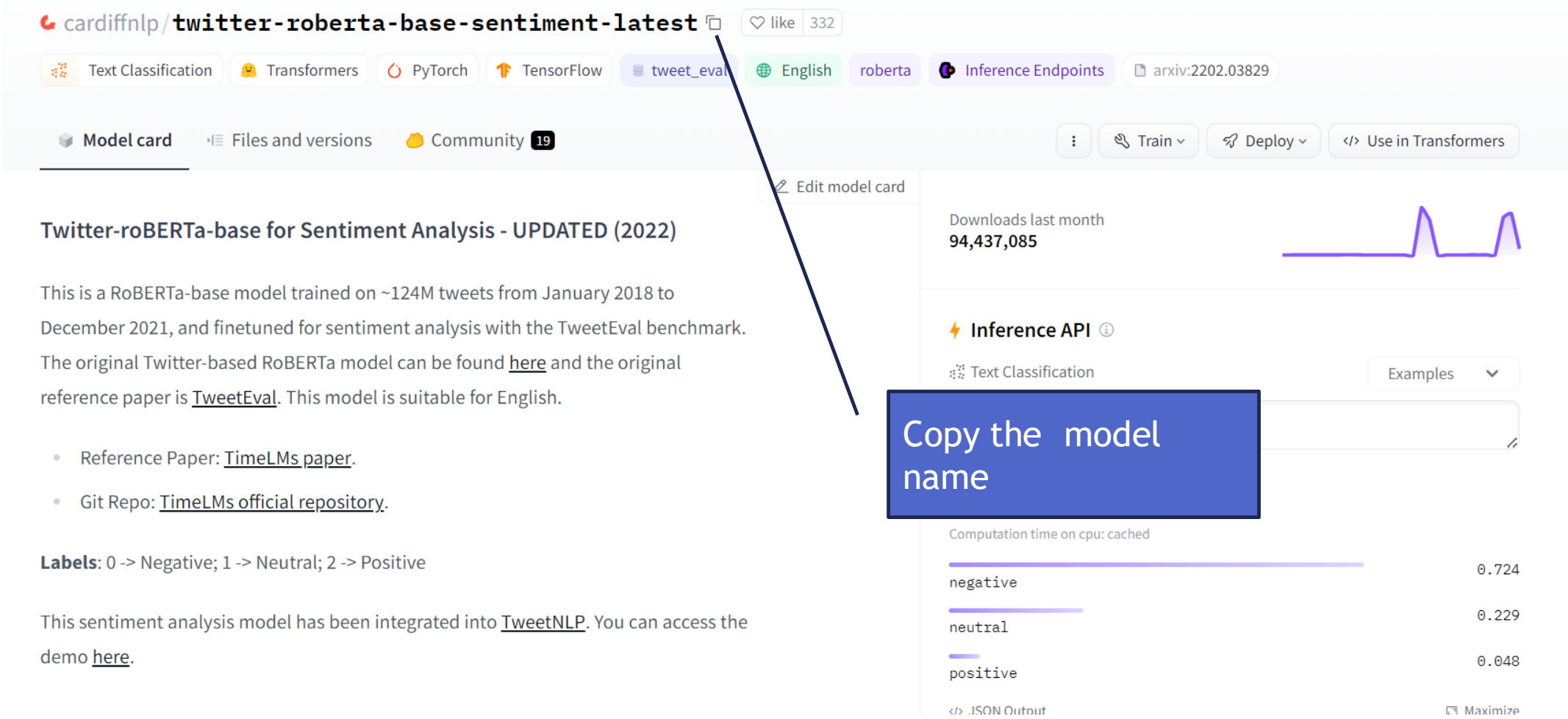



- Select the task
- Sentiment Analysis comes under text classification

Several models available

Sort and choose the most liked or most downloaded or any other model of your choice

# Choose your model



# Choose your model

```
Senti_model_2 = pipeline(task="sentiment-analysis",
                         model="cardiffnlp/twitter-roberta-base-sentiment-latest")
```

```
Senti_model_2("Over heating issue don't by this product camera was good")
```

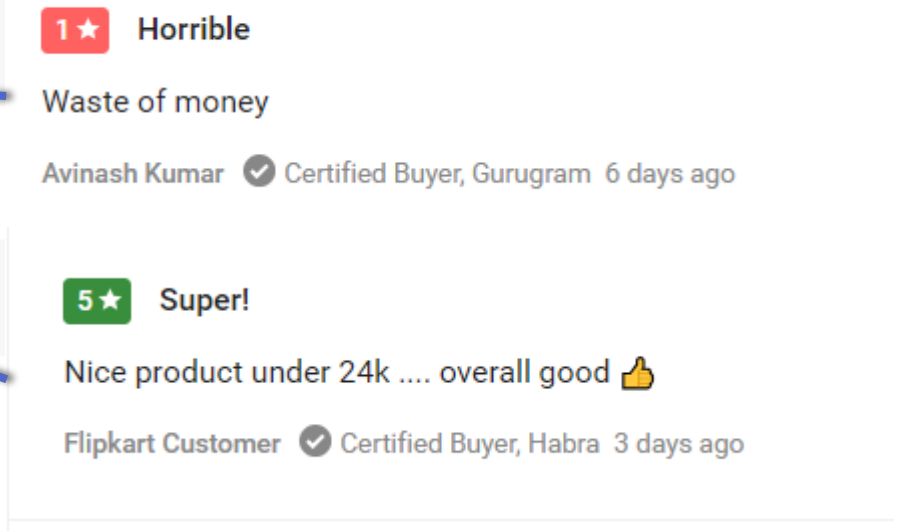
```
[{'label': 'neutral', 'score': 0.4210317134857178}]
```

```
Senti_model_2("Waste of money")
```

```
[{'label': 'negative', 'score': 0.7434294819831848}]
```

```
Senti_model_2("Nice product under 24k .... overall good")
```

```
[{'label': 'positive', 'score': 0.9655607342720032}]
```



# Zero shot prediction

- Zero-shot prediction means asking the model to understand and carry out a task without having been given any specific examples of how to do it during its training.
- It depends on the LLM's broad understanding of language and concepts to make educated guesses.
- **Example:** If an LLM is given a task to classify customer reviews into categories like "service", "quality", "price", without ever being trained on these specific categories, it uses its general knowledge to infer which category a review belongs to based on the content of the review.

# Using this model on your dataset

```
import pandas as pd
user_review_data=pd.read_csv("https://raw.githubuser
user_review_data=user_review_data.sample(50)
user_review_data["Review"]
```

```
23      I have yet to run this new battery below two b...
917                    Leopard Print is wonderfully wild!.
243     No additional ear gels provided, and no instru...
1049    My side Greek salad with the Greek dressing wa...
235                     One of my favorite purchases ever.
1245    -Drinks took close to 30 minutes to come out a...
348     This little device has transformed my organiza...
542     Perhaps my phone is defective, but people cann...
595     What possesed me to get this junk, I have no i...
1708    Service is quick and even "to go" orders are j...
304     Everything worked on the first try.The device ...
325                                 good item, low price.
489     This results in the phone being either stuck a...
949                    I had to purchase a different case.
87             The construction of the headsets is poor
```

Actual Reviews Data

# Sentiment Analysis on a Data frame

```
user_review_data["Predicted_Sentiment"] = user_review_data["Review"].apply(lambda x: Senti_model_2(x)[0]["label"])
user_review_data
```

| | Id | Review | Sentiment | Predicted_Sentiment |
|---|---|---|---|---|
| **0** | 1 | So there is no way for me to plug it in here i... | 0 | negative |
| **1** | 2 | Good case, Excellent value. | 1 | positive |
| **2** | 3 | Great for the jawbone. | 1 | positive |
| **3** | 4 | Tied to charger for conversations lasting more... | 0 | negative |
| **4** | 5 | The mic is great. | 1 | positive |
| **...** | ... | ... | ... | ... |
| **1995** | 1996 | I think food should have flavor and texture an... | 0 | negative |
| **1996** | 1997 | Appetite instantly gone. | 0 | negative |

✓ 1m 5s completed at 10:42 AM

Prediction takes a little longer of we load the model on CPU

# Load the model on GPU

```
Senti_model_2_gpu = pipeline(task="sentiment-analysis",
                             model="cardiffnlp/twitter-roberta-base-sentiment-latest",
                             device="cuda")
```

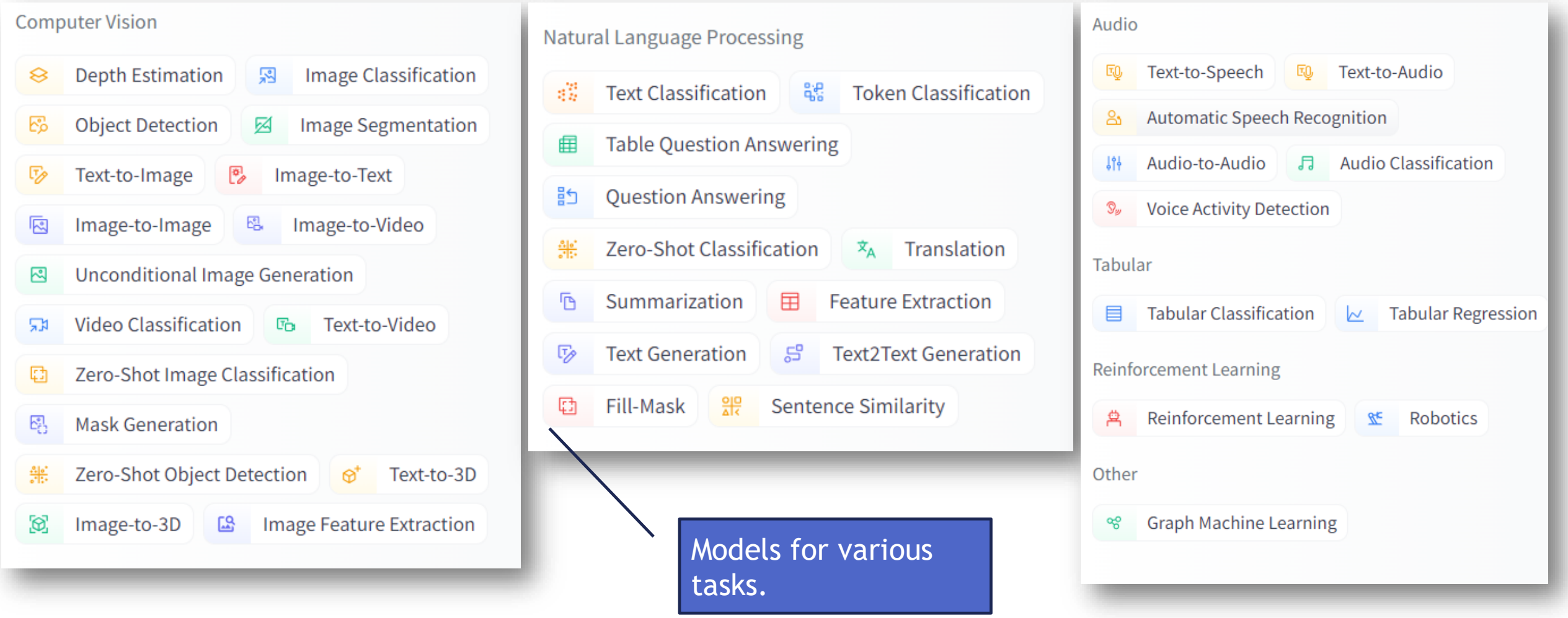
| | Id | Review | Sentiment | Predicted_Sentiment |
|---|---|---|---|---|
| **0** | 1 | So there is no way for me to plug it in here i... | 0 | negative |
| **1** | 2 | Good case, Excellent value. | 1 | positive |
| **2** | 3 | Great for the jawbone. | 1 | positive |
| **3** | 4 | Tied to charger for conversations lasting more... | 0 | negative |
| **4** | 5 | The mic is great. | 1 | positive |
| **...** | ... | ... | ... | ... |
| **1995** | 1996 | I think food should have flavor and texture an... | 0 | negative |
| **1996** | 1997 | Appetite instantly gone. | 0 | negative |
| **1997** | 1998 | Overall I was not impressed and would not go b... | 0 | negative |

✓ 20s completed at 10:40 AM

device="cuda" for GPU

Takes less time to complete it

# Other models on Hugging Face




Models for various tasks.

- ♪ **"audio-classification"**: Returns a **AudioClassificationPipeline**.
- 🗣□ **"automatic-speech-recognition"**: Returns a **AutomaticSpeechRecognitionPipeline**.
- 💬 **"conversational"**: Returns a **ConversationalPipeline**.
- 📏 **"depth-estimation"**: Returns a **DepthEstimationPipeline**.
- 🕮 **"document-question-answering"**: Returns a **DocumentQuestionAnsweringPipeline**.
- 🔍 **"feature-extraction"**: Returns a **FeatureExtractionPipeline**.
- 🎭 **"fill-mask"**: Returns a **FillMaskPipeline**.
- 🖼□ **"image-classification"**: Returns a **ImageClassificationPipeline**.
- 🖼□→🔍 **"image-feature-extraction"**: Returns an **ImageFeatureExtractionPipeline**.
- 🖼□□ **"image-segmentation"**: Returns a **ImageSegmentationPipeline**.
- 🖼□→🖼□ **"image-to-image"**: Returns a **ImageToImagePipeline**.
- 🖼□→📝 **"image-to-text"**: Returns a **ImageToTextPipeline**.
- 🎭 **"mask-generation"**: Returns a **MaskGenerationPipeline**.
- 👤□♂ **"object-detection"**: Returns a **ObjectDetectionPipeline**.

- Some of the currently accepted tasks
- More on next page

- "question-answering": Returns a **QuestionAnsweringPipeline**.
- "summarization": Returns a **SummarizationPipeline**.
- "table-question-answering": Returns a **TableQuestionAnsweringPipeline**.
- "text2text-generation": Returns a **Text2TextGenerationPipeline**.
- "text-classification" (alias "sentiment-analysis"): Returns a **TextClassificationPipeline**.
- "text-generation": Returns a **TextGenerationPipeline**.
- "text-to-audio" (alias "text-to-speech"): Returns a **TextToAudioPipeline**.
- "token-classification" (alias "ner"): Returns a **TokenClassificationPipeline**.
- "translation": Returns a **TranslationPipeline**.
- "translation_xx_to_yy": Returns a **TranslationPipeline**.
- "video-classification": Returns a **VideoClassificationPipeline**.
- "visual-question-answering": Returns a **VisualQuestionAnsweringPipeline**.
- "zero-shot-classification": Returns a **ZeroShotClassificationPipeline**.
- "zero-shot-image-classification": Returns a **ZeroShotImageClassificationPipeline**.
- "zero-shot-audio-classification": Returns a **ZeroShotAudioClassificationPipeline**.
- "zero-shot-object-detection": Returns a **ZeroShotObjectDetectionPipeline**.

We are interested in these NLP based tasks

Python 3 Google Compute Engine backend (GPU)
Showing resources from 10:18 AM to 12:01 PM

System RAM
6.8 / 51.0 GB

GPU RAM
5.8 / 15.0 GB

Disk
28.0 / 201.2 GB

Clear cache in GPU

```
import torch
torch.cuda.empty_cache()
```

Python 3 Google Compute Engine backend (GPU)
Showing resources from 10:18 AM to 12:02 PM

System RAM
6.9 / 51.0 GB

GPU RAM
1.1 / 15.0 GB

Disk
28.0 / 201.2 GB

# Language Translation Model

```
translator_model = pipeline(task="translation_en_to_fr",
                            model="google-t5/t5-small",
                            device="cuda")
translator_model("Good bye")
```

```
[{'translation_text': 'Bon droit'}]
```

# Q&A Model

```
qa_model = pipeline(task="question-answering",
                    model="deepset/roberta-base-squad2",
                    device="cuda")
```

Our model

```
#Importing computer_scientists.txt document from github
!wget https://raw.githubusercontent.com/venkatareddykonasani/D
document=open("computer_scientists.txt").read()
```

Input Document

# Q&A based on a document

```
qa_model({'question':"What did Yann LeCun contribute?",
          'context':document})
```

```
{'score': 0.24149088561534882,
 'start': 330,
 'end': 377,
 'answer': 'Revolutionized AI for image and text processing'}
```

```
qa_model({'question':"Who is the father of deep learning?",
          'context':document})
```

```
{'score': 0.7754606008529663,
 'start': 1421,
 'end': 1444,
 'answer': 'Geoffrey Everest Hinton'}
```

# NER Model

```
ner_model = pipeline(task="ner",
                     model="dslim/bert-base-NER",
                     device="cuda",
                     aggregation_strategy="simple")
#aggregation_strategy ="Simple" ; simplifies the output and makes it easy to read
```

# NER Model

```
sample_doc="""
Hello,
  I, John Smith, a member of the Tech Innovators team, would like to schedule a meeting with you,
  Mary Johnson, from the Quantum Solutions group, on Tuesday, February 8th, 2024, at 10:00 AM.
  We can meet at your office in San Francisco or, if more convenient, at the Cafe Bella in New York City.
  Please let me know if this date and time work for you.
"""
```

NER output

```
entities = ner_model(sample_doc)
print(entities)
```

```
[{'entity_group': 'PER', 'score': 0.9994373, 'word': 'John Smith', 'start': 13, 'end': 23}, {'entity_group': 'ORG', 'score':
0.9968871, 'word': 'Tech Innovators', 'start': 41, 'end': 56}, {'entity_group': 'PER', 'score': 0.99906284, 'word': 'Mary
Johnson', 'start': 108, 'end': 120}, {'entity_group': 'ORG', 'score': 0.9988469, 'word': 'Quantum Solutions', 'start': 131,
'end': 148}, {'entity_group': 'LOC', 'score': 0.9993695, 'word': 'San Francisco', 'start': 233, 'end': 246},
{'entity_group': 'ORG', 'score': 0.63540447, 'word': 'Cafe Bella', 'start': 278, 'end': 288}, {'entity_group': 'LOC',
'score': 0.9994677, 'word': 'New York City', 'start': 292, 'end': 305}]
```

# NER Model

```
# Convert the above output into a dataframe and print it with the entity name
NER_result = pd.DataFrame(entities, columns=["word", "entity_group"])

# Print the DataFrame
print(NER_result)
```

```
                word entity_group
0         John Smith          PER
1    Tech Innovators          ORG
2       Mary Johnson          PER
3  Quantum Solutions          ORG
4      San Francisco          LOC
5         Cafe Bella          ORG
6      New York City          LOC
```

Output stored in a data frame

# Text Summarization Model

```
summarizer_model = pipeline(task="summarization",
                            model="google/pegasus-xsum",
                            device="cuda")
```

```
Book_essay = """
The 7 Habits of Highly Effective People" is a timeless self-help book by Ste
Covey's philosophy centers on the idea that true success is achieved by alig
The next three habits delve into the concept of interdependence, emphasizing
The seventh habit, "Sharpen the Saw," encourages continuous self-renewal and
Throughout the book, Covey provides practical advice and real-life examples
"""
```

```
print(summarizer_model(Book_essay, max_length=120, min_length=30))
```

```
[{'summary_text': '"The 7 Habits of Highly Effective People" is a timeless self-help book by Stephen R. Covey that offers holistic approach to personal and professional effectiveness.'}]
```

# Text Generation Model

```
text_generator_model = pipeline(task="text-generation",
                                model="gpt2",
                                device="cuda")
```

```
# Generate text starting with the given prompt
text_result = text_generator_model("The best way to start a presentation is")
print(text_result)
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
[{'generated_text': 'The best way to start a presentation is to find the most exciting thing about any subject imaginable—
how exciting a subject it is, and the value it creates," says Mark Bivens, a researcher as well as a research associate at
the Massachusetts Institute'}]
```

# pipeline() function

Raw input

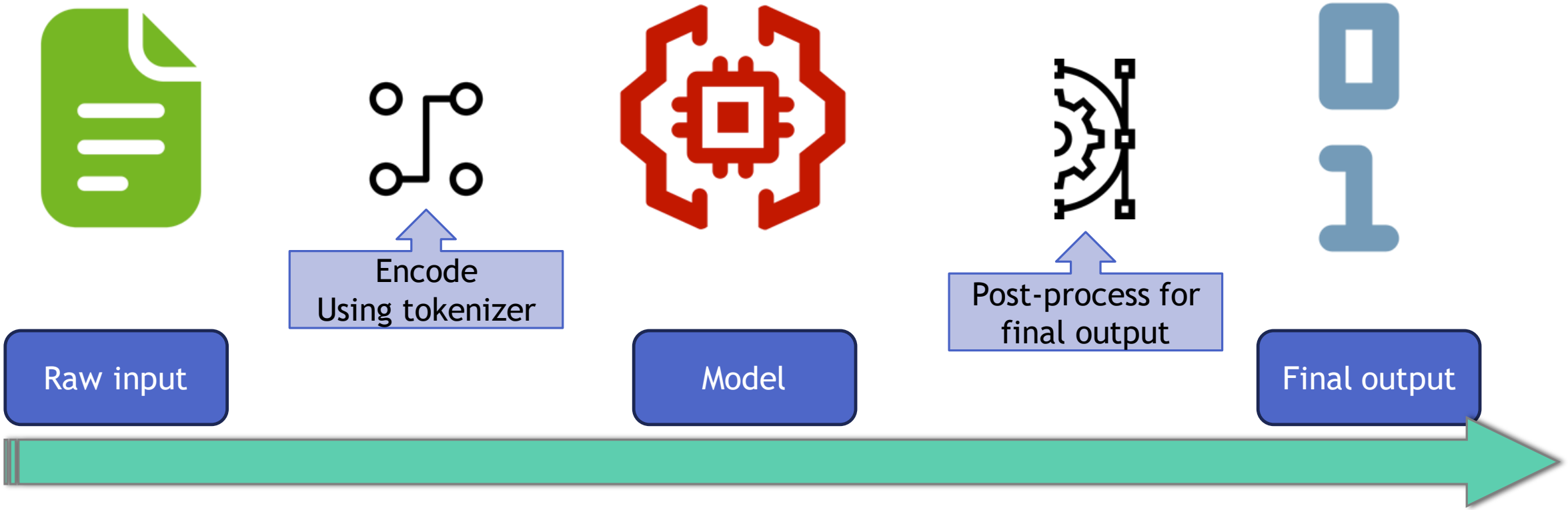
Model

Final output

```
Senti_model_2 = pipeline(task="sentiment-analysis",
                         model="cardiffnlp/twitter-roberta-base-sentiment-latest")
```

```
Senti_model_2("Over heating issue don't by this product camera was good")
```

# Without pipeline() function




- We have to convert raw input text into encoded values using a tokenizer
- Get the output from the model. The output will be in the form of tensors. We need to process it to display the original classes

# Without pipeline() function

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification

tokenizer = AutoTokenizer.from_pretrained("cardiffnlp/twitter-roberta-base-sentiment")

model = AutoModelForSequenceClassification.from_pretrained("cardiffnlp/twitter-roberta-base-sentiment")
```

- Every model has its own tokenizer function.
- Auto tokenizer and Auto model functions work for almost all the models

# Without pipeline() function

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification

tokenizer = AutoTokenizer.from_pretrained("cardiffnlp/twitter-roberta-base-sentiment")

model = AutoModelForSequenceClassification.from_pretrained("cardiffnlp/twitter-roberta-base-sentiment")
```

```
import numpy as np
raw_text = "This is a great book"
encoded_input = tokenizer(raw_text, return_tensor
output = model(**encoded_input)
logits = output.logits.detach().numpy()
y_pred = np.argmax(logits)
y_pred
```

```
2
```

- Model output will be in the form of tensors
- This post processing step will extract the logits from the output
- We use those logits to decide our final predicted class

- **Labels**: 0 -> Negative; 1 -> Neutral; 2 -> Positive

# Finetuning the models on our custom data

# Transfer learning from Hugging Face Models

- So far, we've been using the model just as it was originally trained on.
- The model might NOT have learned from data that’s exactly similar to ours before.
- Now, let's learn how to adjust an LLM model that was already trained, to work better with our data.

# What happens if we don’t fine tune?

- The LLM might find it hard to figure out and sort the categories in our special data.
- We can use the LLM's ability to think and solve problems, but we shouldn't depend only on its Zero-shot prediction(first-time guesses) .
- For instance, in tasks like text classification, LLMs usually look for positive or negative sentiment.
- But what if all we have are bad complaints, yet they're about different kinds or departments?
- How do we help the LLM understand that our data isn't like the usual sentiment analysis data it sees?
- By finetuning and showing some examples to the existing LLM

# The Problem Statement and Dataset


- https://www.consumerfinance.gov/
- The Consumer Financial Protection Bureau (CFPB) acts as a mediator between financial institutions and consumers, facilitating dispute resolution when complaints arise.
- To improve efficiency and accuracy in handling customer complaints, they would like to automatically classify and route complaints to the appropriate teams based on their content and associated financial products.
- https://www.kaggle.com/datasets/adhamelkomy/bank-customer-complaint-analysis/data

# About the dataset

```
!wget https://github.com/venkatareddykonasani/Datasets/raw/maste
!unzip -o complaints_v2.zip
complaints_data = pd.read_csv("/content/complaints_v2.csv")
complaints_data.head()
```

| product | text | label |
|---|---|---|
| credit_card | purchase order day shipping amount receive pro... | 1 |
| credit_card | forwarded message date tue subject please inve... | 1 |
| retail_banking | forwarded message cc sent friday pdt subject f... | 1 |
| credit_reporting | payment history missing credit report speciali... | 0 |
| credit_reporting | payment history missing credit report made mis... | 0 |

- Bank customers complaints data
- Customer send email complaints for various products
- we are broadly classifying everything as “credit reporting” and “other complaints “

# Lets try Zero Shot Prediction without finetuning

```
from transformers import pipeline
distilbert_model = pipeline(task="text-classification",
                            model="distilbert-base-uncased",
                            device="cuda",
                            )
```

- The distil-bert model is lightweight
- We can finetune it in our system
- Remaining models are heavy and we may take a ,lot of execution time.

# Prediction

- Take a sample dataset 100 -1000
- Consider only the initial 350 words in a complaint

```
# Sampling 1000 random entries from the complaints data for analysis
sample_data = complaints_data.sample(1000, random_state=42)

# Trimming each text entry to the first 350 words to reduce processing time and memory usage
sample_data["text"] = sample_data["text"].apply(lambda x: " ".join(x.split()[:350]))

# Applying the DistilBERT model to predict the category of each complaint in the sample
sample_data["bert_predicted"] = sample_data["text"].apply(lambda x: distilbert_model(x)[0]["label"])

# Extracting the numerical part of the predicted category (assuming the label is something like 'LABEL_0')
sample_data["bert_predicted_num"] = sample_data["bert_predicted"].apply(lambda x: x[-1])
```

Prediction for each row

# Accuracy of Zero Shot Prediction

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(sample_data["label"], sample_data["bert_predicted_num"])
print(cm)
accuracy=cm.diagonal().sum()/cm.sum()
print(accuracy)
```

```
[[  0 544]
 [  1 455]]
0.455
```

Almost all the records are predicted as class-1

# Lets finetune on our data

- Steps in finetuning
  1. Import your dataset and pre-process the data and convert it into hugging face friendly datasets
  2. Tokenize and pad the dataset
  3. Define the model and number of labels
  4. Define the training arguments like model output directory
  5. Finetuning step - Train the model by supplying the model and tokenizer
  6. Ultimately utilizing the refined model for making predictions

# Install and Import the packages

```
# Quietly update and install the accelerate package
!pip -q install accelerate -U
# Quietly install the transformers library along with its PyTorch dependencies
!pip -q install transformers[torch]
# Quietly install the Hugging Face datasets library
!pip -q install datasets

# Importing necessary libraries and classes for model training and data handling
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification, Trainer, TrainingArguments
from transformers import Trainer, TrainingArguments  # Trainer and TrainingArguments are imported twice, which
from datasets import load_dataset, DatasetDict, ClassLabel, Dataset
import pandas as pd  # For data manipulation and analysis
from sklearn.model_selection import train_test_split  # For splitting data into training and testing sets
import torch  # PyTorch library for deep learning applications
```

# Hugging face friendly datasets

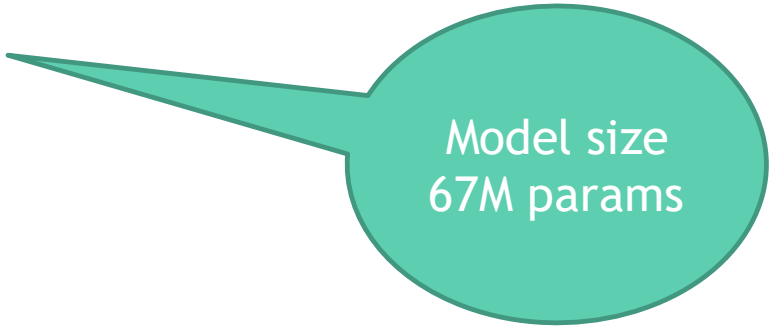
```
# Convert the pandas DataFrame `sample_data` to a Hugging Face `Dataset`
Sample_data = Dataset.from_pandas(sample_data)

# Split the dataset into training and testing sets
# Using an 80-20 split for training and testing, respectively
train_test_split = Sample_data.train_test_split(test_size=0.2)  # 80% training, 20% testing

# Creating a DatasetDict to organize the train and test splits
dataset = DatasetDict({
    'train': train_test_split['train'],
    'test': train_test_split['test']
})
```

# Tokenize and pad your data

```
# Load the tokenizer for the 'distilbert-base-uncased' model
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')

# Setting up padding token to be the same as the EOS (end of sequence) token
# This aligns padding behavior with DistilBERT's expected inputs
tokenizer.pad_token = tokenizer.eos_token
tokenizer.pad_token_id = tokenizer.eos_token_id

# Adding a special pad token ('[PAD]') to the tokenizer
# This step might not be necessary since pad_token is already set to eos_token, but it explicitly ensures '[PA
tokenizer.add_special_tokens({'pad_token': '[PAD]'})

# Define a function to tokenize the input texts
def tokenize_function(examples):
    # Tokenizes the examples text, ensures they are padded to a maximum length of 512 tokens, and truncated if
    return tokenizer(examples["text"], padding="max_length", truncation=True, max_length=512)

# Apply the tokenize function to all examples in the dataset
# The 'batched=True' option processes the texts in batches, making this operation faster
tokenized_datasets = dataset.map(tokenize_function, batched=True)
```

```python
# Initialize the DistilBERT model for sequence classification with the 'distilbert-base-uncased' pre-trained m
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased',
                                                            num_labels=2,  # Adjust this based on the number o
                                                            pad_token_id=tokenizer.eos_token_id)  # Set the pa
```

```
DistilBertForSequenceClassification(
  (distilbert): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0-5): 6 x TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
  )
  (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
```



# Define the model and the training arguments

```
# Initialize the DistilBERT model for sequence classification with the 'distilbert-base-uncased' pre-trained m
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased',
                                                             num_labels=2,  # Adjust this based on the number o
                                                             pad_token_id=tokenizer.eos_token_id)  # Set the pa

# Set up the training arguments specifying the directory for saving results, the number of training epochs, an
training_args = TrainingArguments(
    output_dir="./results_bert_custom",  # Directory where the training results will be saved
    num_train_epochs=1,  # Number of epochs to train for
    logging_dir="./logs_bert_custom",  # Directory where the training logs will be saved
    evaluation_strategy="epoch",  # Evaluate the model at the end of each epoch
)
```

# Train the model

```
# Initialize the Trainer with the model, training arguments, and the train and evaluation datasets
trainer = Trainer(
    model=model,  # The pre-initialized DistilBERT model
    args=training_args,  # Training arguments specifying the training setup
    train_dataset=tokenized_datasets['train'],  # The tokenized training dataset
    eval_dataset=tokenized_datasets['test'],  # The tokenized evaluation dataset
)

# Start the training process
trainer.train()
```

This is the Fine-tuning of the BERT model to align with our updated dataset by modifying the weight

This training takes a lot of time

# Save the model

```
# Define the directory where you want to save your model and tokenizer
model_dir = "./distilbert_finetuned"

# Save the model
model.save_pretrained(model_dir)

# Save the tokenizer
tokenizer.save_pretrained(model_dir)

#Save the model with
trainer.save_model('Distilbert_CustomModel_10K')
```

# Make the predictions using the finetuned model

```
# Define a function for making predictions with the finetuned model
def make_prediction(text):
    # Prepare the input text for the model
    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=512)
    # Move the inputs to the GPU
    inputs = inputs.to(torch.device("cuda:0"))
    # Get model predictions
    outputs = model(**inputs)
    # Extract the predicted class (the one with the highest score) from the logits
    predictions = outputs.logits.argmax(-1)
    # Move the predictions back to CPU and convert to numpy for easy handling
    predictions = predictions.detach().cpu().numpy()
    # Return the predicted class
    return predictions

# Apply the prediction function to the 'text' column of the sample_data DataFrame
# Note: If running this line results in performance issues or out-of-memory errors, consider applying predicti
sample_data["finetuned_predicted"] = sample_data["text"].apply(lambda x: make_prediction(str(x))[0])
```

If this doesn't work, use my pre-trained model

# Loading a pre-built model and making prediction



```
#Code to donwload the distilbert model
!gdown --id 1785J3ir19RaZP3ebbFvWUX88PMaBouro -O distilbert_finetuned_V1.zip
!unzip -o -j distilbert_finetuned_V1.zip -d distilbert_finetuned_V1

model_v1 = DistilBertForSequenceClassification.from_pretrained('/content/distilbert_finetuned_V1')
model_v1.to("cuda:0")
```

# Accuracy with the new finetuned model

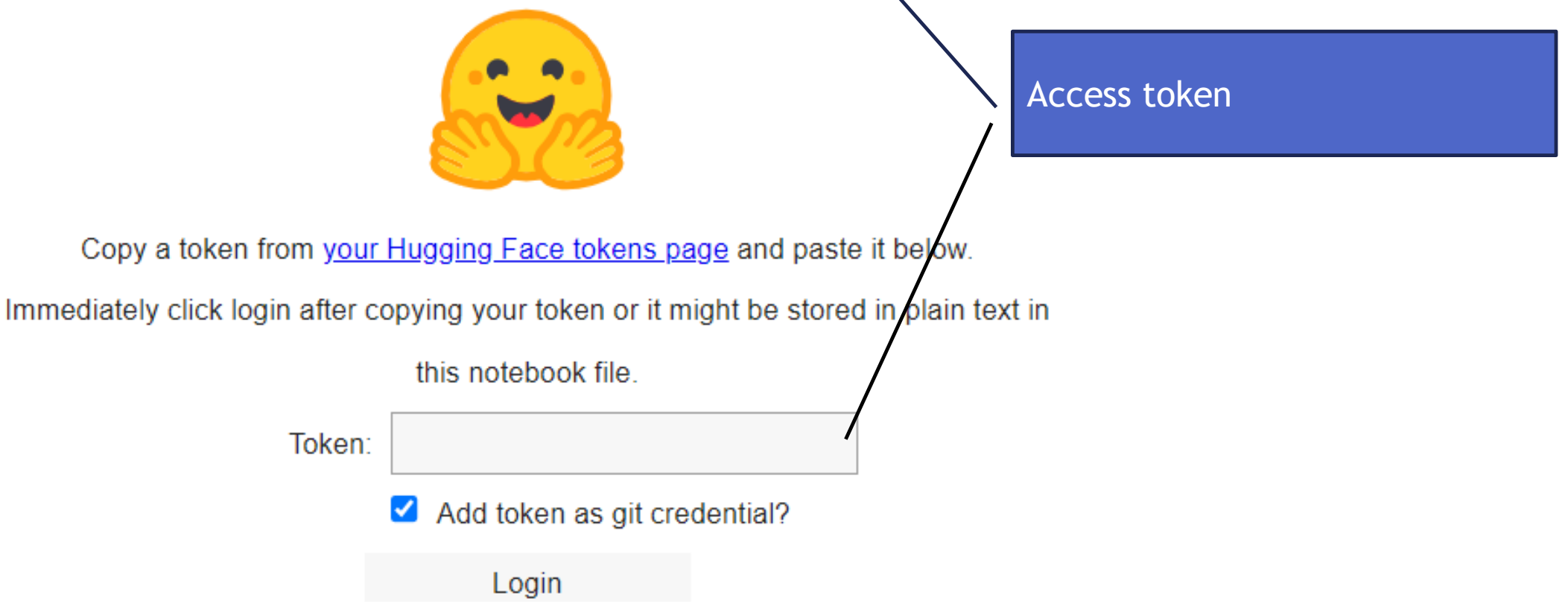
```
from sklearn.metrics import confusion_matrix
# Create the confusion matrix
cm1 = confusion_matrix(sample_data["label"], sample_data["finetuned_predicted"])
print(cm1)
accuracy1=cm1.diagonal().sum()/cm1.sum()
print(accuracy1)
```

```
[[479  65]
 [ 81 375]]
0.854
```

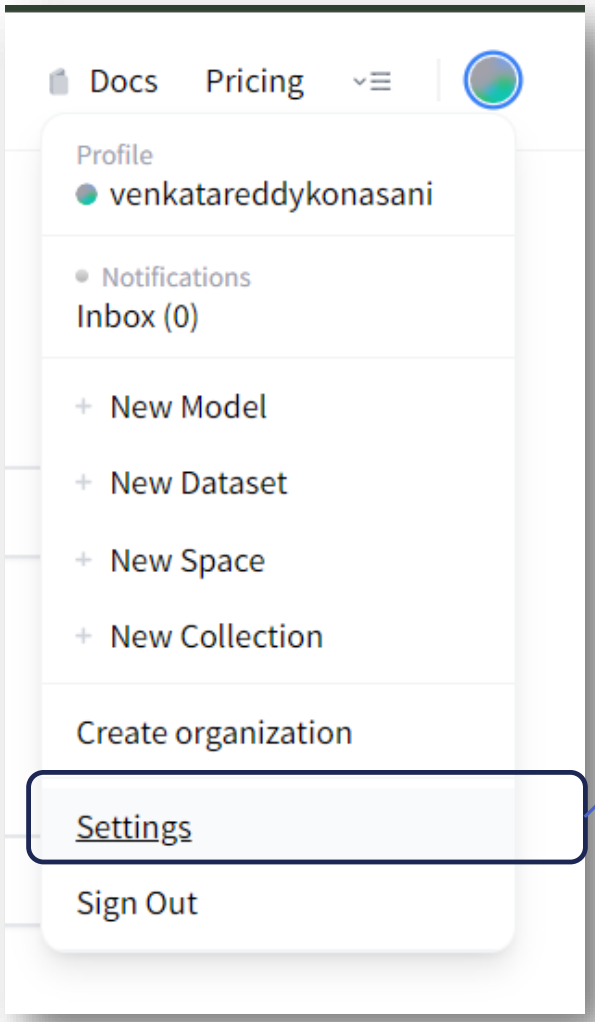
Much better than the previous model

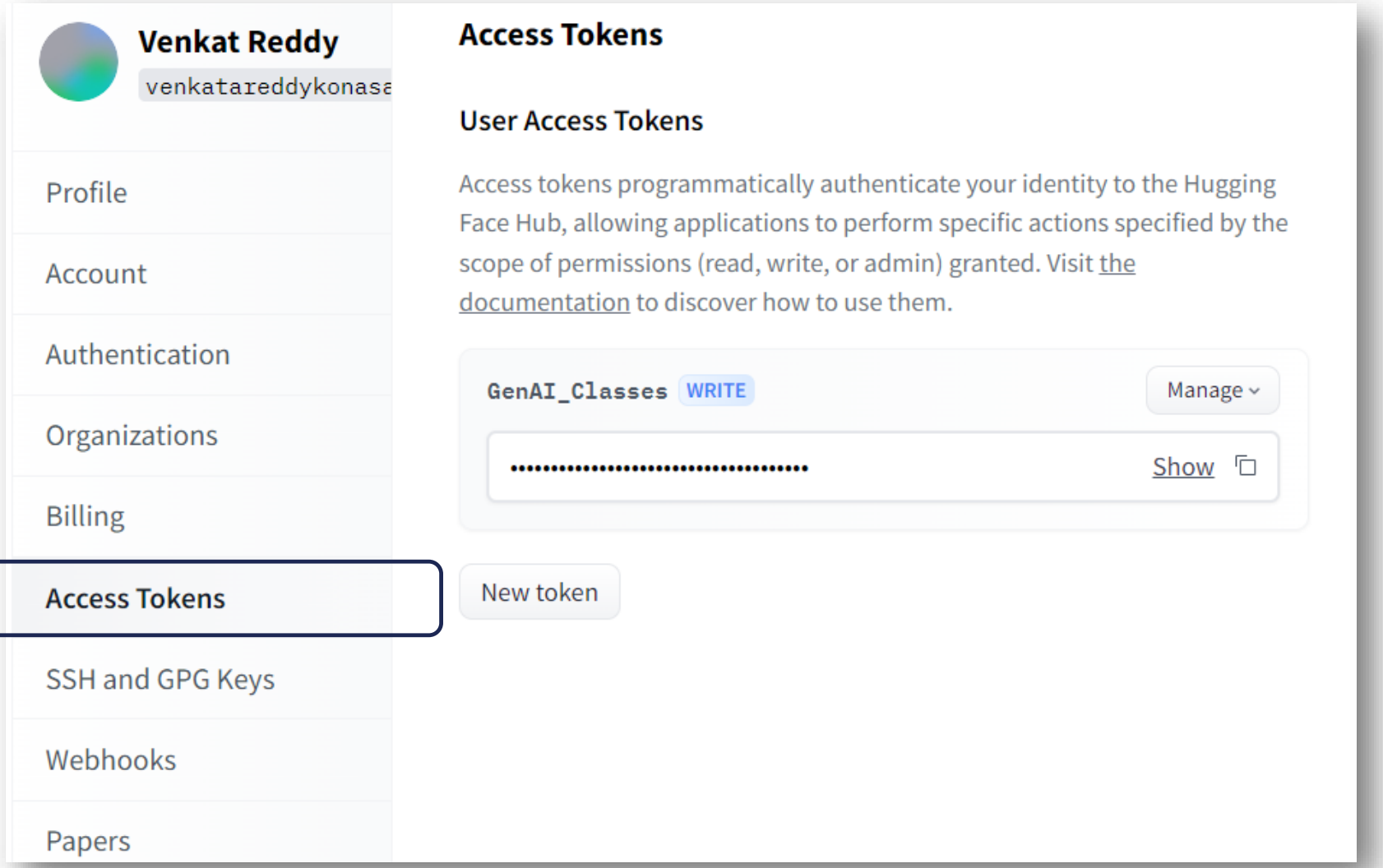
# Saving the Model on Hugging Face hub

```
from huggingface_hub import notebook_login
notebook_login()
#To get Auth token: Profile >> Settings >>Access Token
```



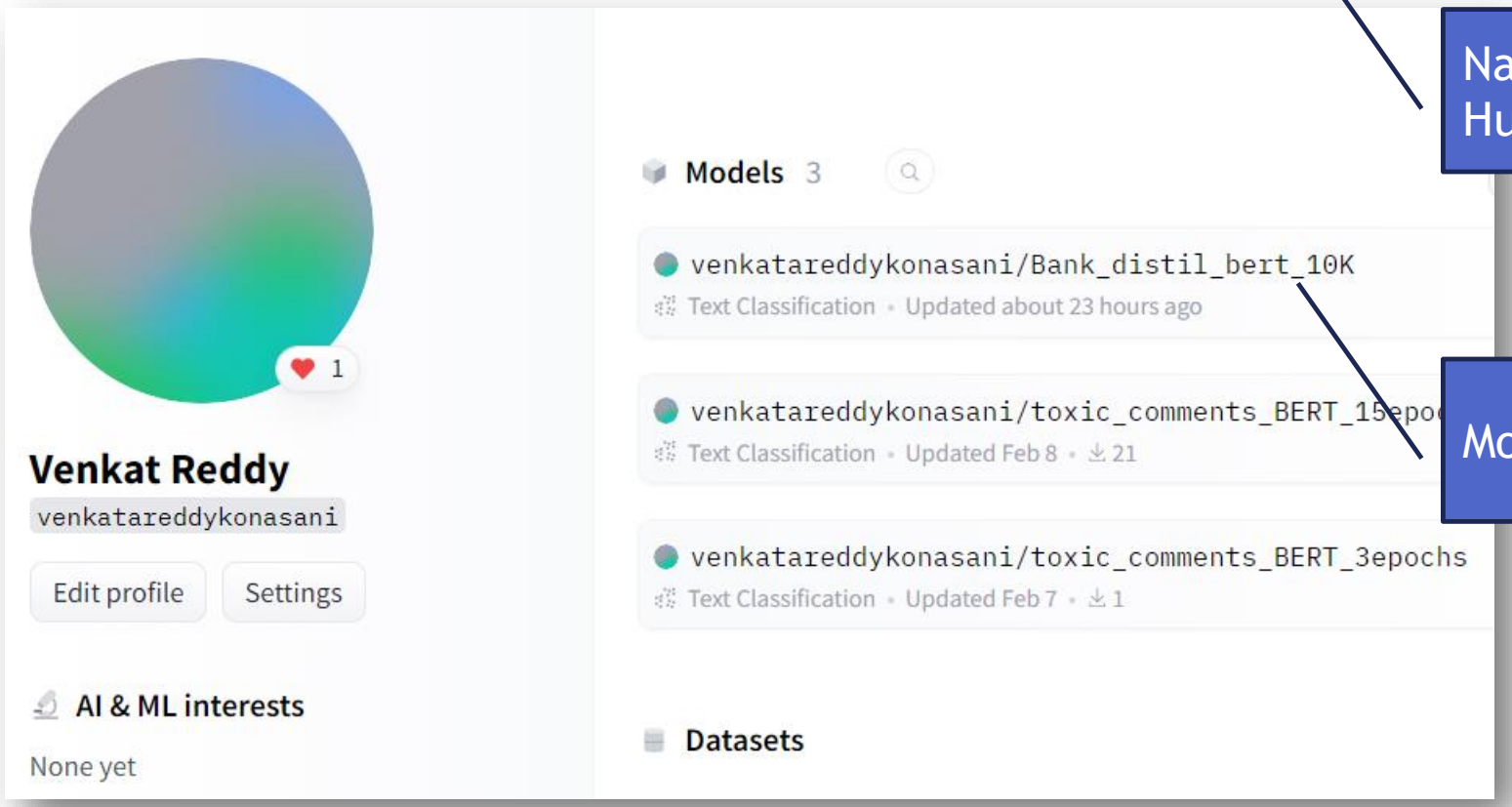
# Get your API key here



# Push the model to HuggingFace hub

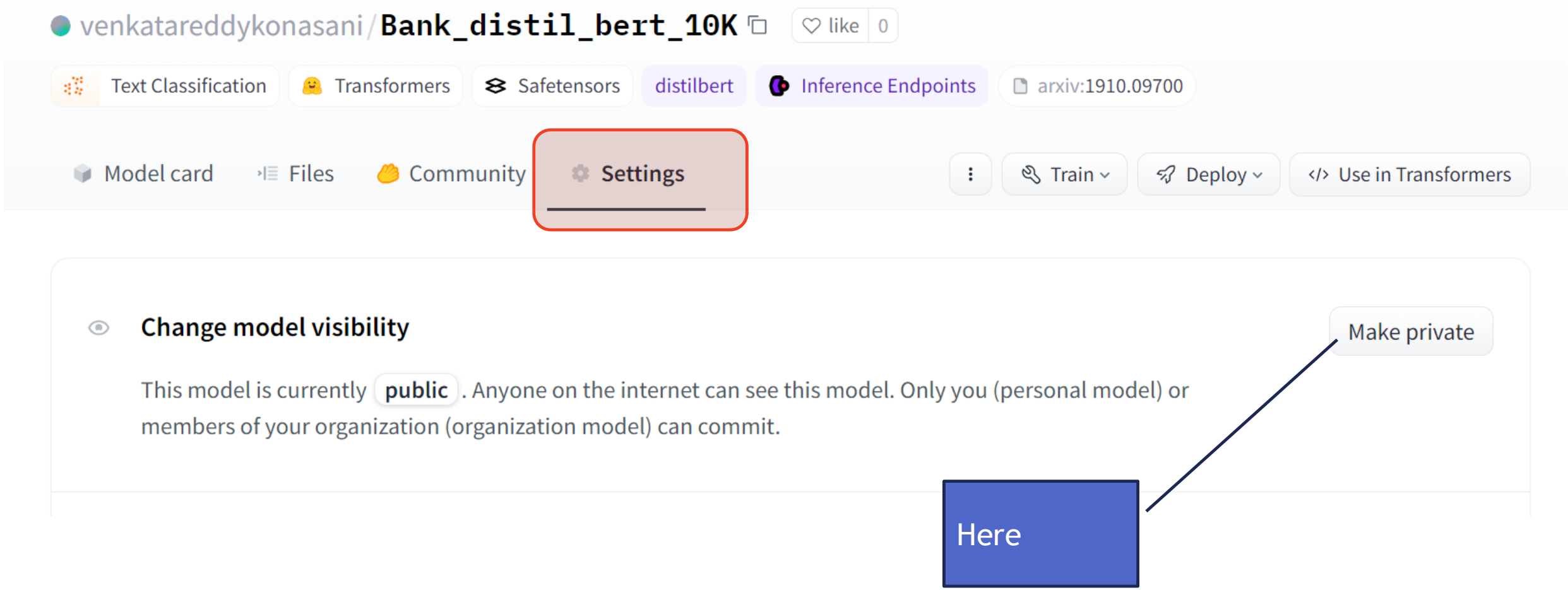
```
model.push_to_hub("venkatareddykonasani/Bank_distil_bert_10K")
```




Name to be appeared on Hugging Face

Model on the Hugging Face hub

# Make the model private



# Loading the model from Hugging Face

```
model=DistilBertForSequenceClassification.from_pretrained("venkatareddykonasani/Bank_distil_bert_10K")
```

```
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification

tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
```

# Creating a WebApp

# Model Deployment on Streamlit

```
%%writefile requirements.txt
streamlit
numpy
pandas
torch
transformers
huggingface_hub
```

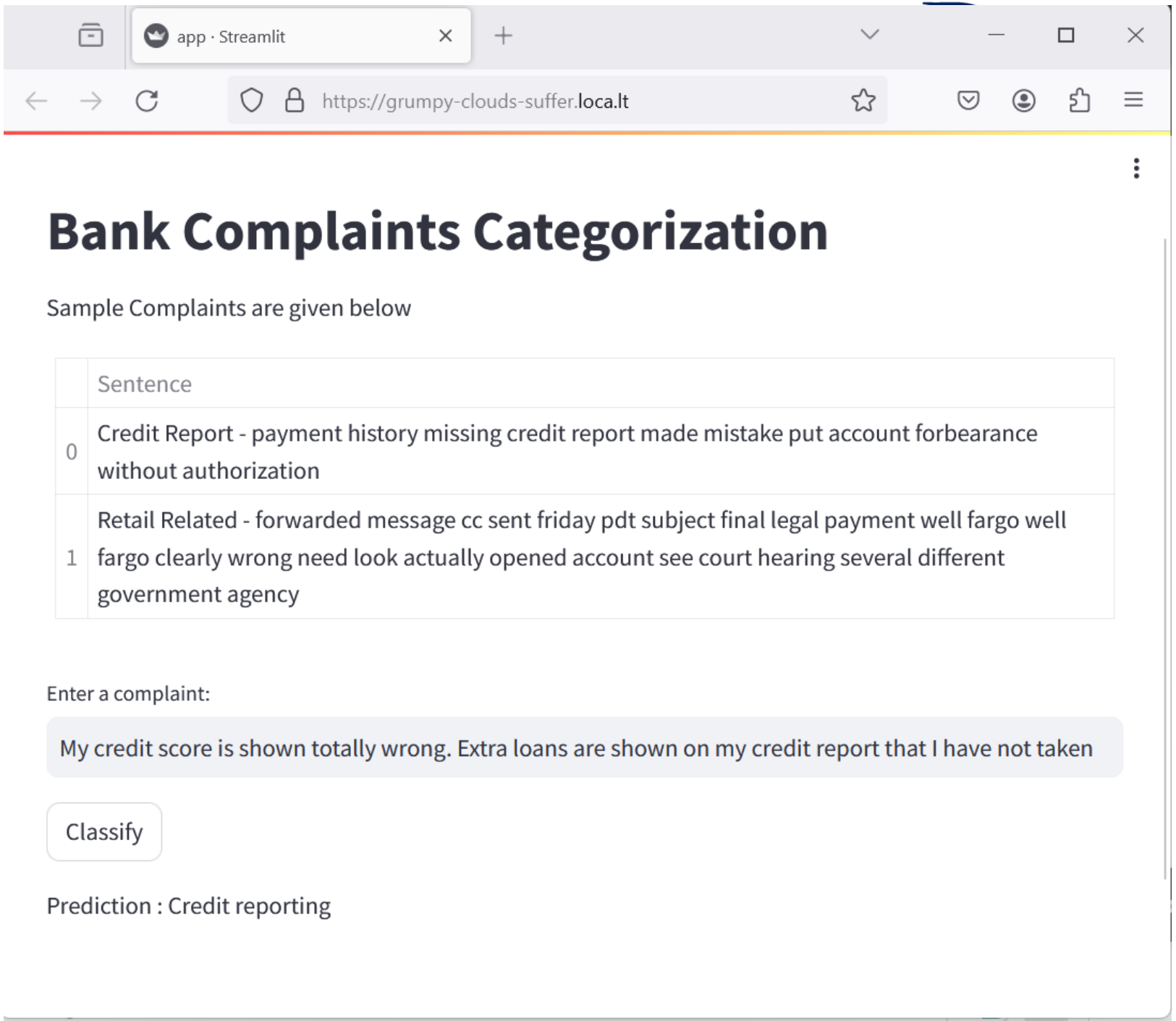
# Model Deployment on Streamlit

```
import streamlit as st
import numpy as np
import pandas as pd
import torch
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification, Trainer,

tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
model = DistilBertForSequenceClassification.from_pretrained('venkatareddykonasani/Bank_dist

st.title("Bank Complaints Categorization")
st.write("Sample Complaints are given below")
Sample_Complaints = [
    {"Sentence": "Credit Report - payment history missing credit report made mistake put ac
    {"Sentence": "Retail Related - forwarded message cc sent friday pdt subject final legal
]
st.table(Sample_Complaints)
user_input = st.text_input("Enter a complaint:")
button=st.button("Classify")
```

# Model Deployment on Streamlit

```
d={
    0: "Credit reporting",
    1: "Mortgage and Others"
}

if user_input and button:
  inputs=tokenizer(user_input, return_tensors="pt")
  outputs=model(**inputs)
  predictions=outputs.logits.argmax(-1)
  predictions=predictions.detach().cpu().numpy()
  print(predictions)
  st.write("Prediction :" , d[predictions[0]])
```

# Final app


app · Streamlit
https://grumpy-clouds-suffer.loca.lt
Bank Complaints Categorization
Sample Complaints are given below
Sentence
0
Credit Report - payment history missing credit report made mistake put account forbearance without authorization
1
Retail Related - forwarded message cc sent friday pdt subject final legal payment well fargo well fargo clearly wrong need look actually opened account see court hearing several different government agency
Enter a complaint:
My credit score is shown totally wrong. Extra loans are shown on my credit report that I have not taken
Classify
Prediction : Credit reporting