Nama : M. Rizky Fadillah
NIM : 1103210259

TUBES MACHINE LEARNING

Task 3 (5 – 9)

5. The Datasets Library

- What if my dataset isn't on the Hub?

```
!pip install datasets evaluate transformers[sentencepiece]

!wget https://github.com/crux82/squad-it/raw/master/SQuAD_it-train.json.gz
!wget https://github.com/crux82/squad-it/raw/master/SQuAD_it-test.json.gz

!gzip -dkv SQuAD_it-*.json.gz

from datasets import load_dataset

squad_it_dataset = load_dataset("json", data_files="SQuAD_it-train.json",
field="data")

squad_it_dataset
```

⇨

```
DatasetDict({
    train: Dataset({
        features: ['title', 'paragraphs'],
        num_rows: 442
    })
})
```

```
squad_it_dataset["train"][0]
```

```
{
    "title": "Terremoto del Sichuan del 2008",
    "paragraphs": [
        {
            "context": "Il terremoto del Sichuan del 2008 o il terremoto...",
            "qas": [
                {
                    "answers": [{"answer_start": 29, "text": "2008"}],
                    "id": "56cdca7862d2951400fa6826",
                    "question": "In quale anno si è verificato il terremoto nel Sichuan?",
                },
                ...
            ],
        },
        ...
    ],
}
```

```
data_files = {"train": "SQuAD_it-train.json", "test": "SQuAD_it-
test.json"}
squad_it_dataset = load_dataset("json", data_files=data_files,
field="data")
squad_it_dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['title', 'paragraphs'],
        num_rows: 442
    })
    test: Dataset({
        features: ['title', 'paragraphs'],
        num_rows: 48
    })
})
```

```
data_files={"train":"SQuAD_ittrain.json.gz","test":"SQuAD_ittest.json.gz"}
squad_it_dataset = load_dataset("json",data_files=data_files,field="data")
```

```
[ ]  url = "https://github.com/crux82/squad-it/raw/master/"
     data_files = {
         "train": url + "SQuAD_it-train.json.gz",
         "test": url + "SQuAD_it-test.json.gz",
     }
     squad_it_dataset = load_dataset("json", data_files=data_files, field="data")
```

Notebook ini menyediakan panduan dasar untuk mempersiapkan dataset yang tidak terdapat di platform Hugging Face. Dengan dataset tersebut, pengguna dapat melanjutkan ke langkah-langkah seperti memuat data, melakukan preprocessing, melatih model, atau mengevaluasi performa model dalam tugas tertentu.

- **Time to slice and dice**

```
!pip install datasets evaluate transformers[sentencepiece]
```

⇨

Kode ini menginstal library yang diperlukan untuk pengembangan proyek NLP, termasuk datasets untuk memuat dan mengelola dataset, evaluate untuk mengevaluasi kinerja model, dan transformers[sentencepiece] untuk bekerja dengan model-model canggih dari Hugging Face yang menggunakan SentencePiece tokenizer, sehingga memudahkan implementasi dan eksperimen dengan berbagai model bahasa alami.

```
!wget "https://archive.ics.uci.edu/ml/machine-learning-
databases/00462/drugsCom_raw.zip"
!unzip drugsCom_raw.zip
```

```
from datasets import load_dataset

data_files = {"train": "drugsComTrain_raw.tsv", "test":
"drugsComTest_raw.tsv"}
# \t is the tab character in Python
drug_dataset = load_dataset("csv", data_files=data_files, delimiter="\t")
```

```
drug_sample = drug_dataset["train"].shuffle(seed=42).select(range(1000))
# Print the first few examples of the dataset
drug_sample[:3]
```

```
{'Unnamed: 0': [87571, 178045, 80482],
 'drugName': ['Naproxen', 'Duloxetine', 'Mobic'],
 'condition': ['Gout, Acute', 'ibromyalgia', 'Inflammatory Conditions'],
 'review': ['"like the previous person mention, I&#039;m a strong believer of aleve, it works
faster for my gout than the prescription meds I take. No more going to the doctor for
refills.....Aleve works!"',
  '"I have taken Cymbalta for about a year and a half for fibromyalgia pain. It is great\r\nas a
pain reducer and an anti-depressant, however, the side effects outweighed \r\nany benefit I got
from it. I had trouble with restlessness, being tired constantly,\r\ndizziness, dry mouth,
numbness and tingling in my feet, and horrible sweating. I am\r\nbeing weaned off of it now. Went
from 60 mg to 30mg and now to 15 mg. I will be\r\noff completely in about a week. The fibro pain
is coming back, but I would rather deal with it than the side effects."',
  '"I have been taking Mobic for over a year with no side effects other than an elevated blood
pressure.  I had severe knee and ankle pain which completely went away after taking Mobic.  I
attempted to stop the medication however pain returned after a few days."'],
 'rating': [9.0, 3.0, 10.0],
 'date': ['September 2, 2015', 'November 7, 2011', 'June 5, 2013'],
 'usefulCount': [36, 13, 128]}
```

```
for split in drug_dataset.keys():
  assertlen(drug_dataset[split])==len(drug_dataset[split].unique("Unnamed:
0"))
```

```
drug_dataset = drug_dataset.rename_column(
    original_column_name="Unnamed: 0", new_column_name="patient_id"
)
drug_dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['patient_id', 'drugName', 'condition', 'review', 'rating', 'date',
'usefulCount'],
        num_rows: 161297
    })
    test: Dataset({
        features: ['patient_id', 'drugName', 'condition', 'review', 'rating', 'date',
'usefulCount'],
        num_rows: 53766
    })
})
```

```
def lowercase_condition(example):
    return {"condition": example["condition"].lower()}


drug_dataset.map(lowercase_condition)
```

```
AttributeError: 'NoneType' object has no attribute 'lower'
```

```
def filter_nones(x):
    return x["condition"] is not None
```

```
(lambda x: x * x)(3)
```

```
9
```

```
(lambda base, height: 0.5 * base * height)(4, 8)
```

```
16.0
```

```
drug_dataset = drug_dataset.filter(lambda x: x["condition"] is not None)
```

```
drug_dataset = drug_dataset.map(lowercase_condition)
# Check that lowercasing worked
drug_dataset["train"]["condition"][:3]
```

```
['left ventricular dysfunction', 'adhd', 'birth control']
```

```
def compute_review_length(example):
    return {"review_length": len(example["review"].split())}
```

```
drug_dataset = drug_dataset.map(compute_review_length)
# Inspect the first training example
drug_dataset["train"][0]
```

```
{'patient_id': 206461,
 'drugName': 'Valsartan',
 'condition': 'left ventricular dysfunction',
 'review': '"It has no side effect, I take it in combination of Bystolic 5 Mg and Fish Oil"',
 'rating': 9.0,
 'date': 'May 20, 2012',
 'usefulCount': 27,
 'review_length': 17}
```

```
drug_dataset["train"].sort("review_length")[:3]
```

```
{'patient_id': [103488, 23627, 20558],
 'drugName': ['Loestrin 21 1 / 20', 'Chlorzoxazone', 'Nucynta'],
 'condition': ['birth control', 'muscle spasm', 'pain'],
 'review': ['"Excellent."', '"useless"', '"ok"'],
 'rating': [10.0, 1.0, 6.0],
 'date': ['November 4, 2008', 'March 24, 2017', 'August 20, 2016'],
 'usefulCount': [5, 2, 10],
 'review_length': [1, 1, 1]}
```

```
drug_dataset = drug_dataset.filter(lambda x: x["review_length"] > 30)
print(drug_dataset.num_rows)
```

```
{'train': 138514, 'test': 46108}
```

```
import html

text = "I&#039;m a transformer called BERT"
html.unescape(text)
```

```
"I'm a transformer called BERT"
```

```
drug_dataset = drug_dataset.map(lambda x: {"review":
html.unescape(x["review"])})
```

```
new_drug_dataset = drug_dataset.map(
    lambda x: {"review": [html.unescape(o) for o in x["review"]]},
batched=True
)
```

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")


def tokenize_function(examples):
    return tokenizer(examples["review"], truncation=True)
```

```
%time tokenized_dataset = drug_dataset.map(tokenize_function,
batched=True)
```

```
slow_tokenizer = AutoTokenizer.from_pretrained("bert-base-cased",
use_fast=False)


def slow_tokenize_function(examples):
    return slow_tokenizer(examples["review"], truncation=True)


tokenized_dataset = drug_dataset.map(slow_tokenize_function, batched=True,
num_proc=8)
```

```
def tokenize_and_split(examples):
    return tokenizer(
        examples["review"],
        truncation=True,
```

```
        max_length=128,
        return_overflowing_tokens=True,
    )
```

```
result = tokenize_and_split(drug_dataset["train"][0])
[len(inp) for inp in result["input_ids"]]
```

```
[128, 49]
```

```
tokenized_dataset = drug_dataset.map(tokenize_and_split, batched=True)
```

```
ArrowInvalid: Column 1 named condition expected length 1463 but got length 1000
```

```
tokenized_dataset = drug_dataset.map(
    tokenize_and_split, batched=True,
remove_columns=drug_dataset["train"].column_names
)
```

```
len(tokenized_dataset["train"]), len(drug_dataset["train"])
```

```
(206772, 138514)
```

```
def tokenize_and_split(examples):
    result = tokenizer(
        examples["review"],
        truncation=True,
        max_length=128,
        return_overflowing_tokens=True,
    )
    # Extract mapping between new and old indices
    sample_map = result.pop("overflow_to_sample_mapping")
    for key, values in examples.items():
        result[key] = [values[i] for i in sample_map]
    return result
```

```
tokenized_dataset = drug_dataset.map(tokenize_and_split, batched=True)
tokenized_dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['attention_mask', 'condition', 'date', 'drugName', 'input_ids', 'patient_id',
'rating', 'review', 'review_length', 'token_type_ids', 'usefulCount'],
        num_rows: 206772
    })
    test: Dataset({
        features: ['attention_mask', 'condition', 'date', 'drugName', 'input_ids', 'patient_id',
'rating', 'review', 'review_length', 'token_type_ids', 'usefulCount'],
        num_rows: 68876
    })
})
```

```
drug_dataset.set_format("pandas")
```

```
drug_dataset["train"][:3]
```

```
train_df = drug_dataset["train"][:]
```

```
frequencies = (
    train_df["condition"]
    .value_counts()
    .to_frame()
    .reset_index()
    .rename(columns={"index": "condition", "condition": "frequency"})
)
frequencies.head()
```

```
from datasets import Dataset

freq_dataset = Dataset.from_pandas(frequencies)
freq_dataset
```

```
Dataset({
    features: ['condition', 'frequency'],
    num_rows: 819
})
```

```
drug_dataset.reset_format()
```

```
drug_dataset_clean =
drug_dataset["train"].train_test_split(train_size=0.8, seed=42)
# Rename the default "test" split to "validation"
drug_dataset_clean["validation"] = drug_dataset_clean.pop("test")
# Add the "test" set to our `DatasetDict`
drug_dataset_clean["test"] = drug_dataset["test"]
```

```
drug_dataset_clean
```

```
DatasetDict({
    train: Dataset({
        features: ['patient_id', 'drugName', 'condition', 'review', 'rating', 'date',
'usefulCount', 'review_length', 'review_clean'],
        num_rows: 110811
    })
    validation: Dataset({
        features: ['patient_id', 'drugName', 'condition', 'review', 'rating', 'date',
'usefulCount', 'review_length', 'review_clean'],
        num_rows: 27703
    })
    test: Dataset({
        features: ['patient_id', 'drugName', 'condition', 'review', 'rating', 'date',
'usefulCount', 'review_length', 'review_clean'],
        num_rows: 46108
    })
})
```

```
drug_dataset_clean.save_to_disk("drug-reviews")
from datasets import load_from_disk

drug_dataset_reloaded = load_from_disk("drug-reviews")
drug_dataset_reloaded
```

```
DatasetDict({
    train: Dataset({
        features: ['patient_id', 'drugName', 'condition', 'review', 'rating', 'date',
'usefulCount', 'review_length'],
        num_rows: 110811
    })
    validation: Dataset({
        features: ['patient_id', 'drugName', 'condition', 'review', 'rating', 'date',
'usefulCount', 'review_length'],
        num_rows: 27703
    })
    test: Dataset({
        features: ['patient_id', 'drugName', 'condition', 'review', 'rating', 'date',
'usefulCount', 'review_length'],
        num_rows: 46108
    })
})
```

```
for split, dataset in drug_dataset_clean.items():
    dataset.to_json(f"drug-reviews-{split}.jsonl")
```

```
!head -n 1 drug-reviews-train.jsonl
```


```
{"patient_id":141780,"drugName":"Escitalopram","condition":"depression","review":"\"I seemed to
experience the regular side effects of LEXAPRO, insomnia, low sex drive, sleepiness during the
day. I am taking it at night because my doctor said if it made me tired to take it at night. I
assumed it would and started out taking it at night. Strange dreams, some pleasant. I was
diagnosed with fibromyalgia. Seems to be helping with the pain. Have had anxiety and depression in
my family, and have tried quite a few other medications that haven't worked. Only have been on it
for two weeks but feel more positive in my mind, want to accomplish more in my life. Hopefully the
side effects will dwindle away, worth it to stick with it from hearing others responses. Great
medication.\"","rating":9.0,"date":"May 29, 2011","usefulCount":10,"review_length":125}
```

```
data_files = {
    "train": "drug-reviews-train.jsonl",
    "validation": "drug-reviews-validation.jsonl",
    "test": "drug-reviews-test.jsonl",
}
drug_dataset_reloaded = load_dataset("json", data_files=data_files)
```

Kesimpulan

Notebook ini dirancang sebagai panduan untuk memanipulasi, memproses, dan mengevaluasi dataset dalam proyek NLP menggunakan pustaka modern seperti Hugging Face Transformers. Dengan langkah awal berupa instalasi pustaka utama seperti datasets, evaluate, dan transformers[sentencepiece], notebook ini mempersiapkan lingkungan kerja yang mendukung pengolahan data dan implementasi model NLP canggih. Fokus utama adalah membantu pengguna memahami dan menerapkan teknik preprocessing dataset, seperti memotong atau menyusun ulang data, serta menggunakan model transformer untuk melatih atau mengevaluasi performa model secara efektif. Notebook ini memberikan dasar yang kuat bagi pengembangan proyek machine learning berbasis data dengan alat yang canggih dan mudah diakses.

- **Big data? 🤗 Datasets to the rescue!**

```
!pip install datasets evaluate transformers[sentencepiece]
```

⇨

Kode ini menginstal library yang diperlukan untuk pengembangan proyek NLP, termasuk datasets untuk memuat dan mengelola dataset, evaluate untuk mengevaluasi kinerja model, dan transformers[sentencepiece] untuk bekerja dengan model-model canggih dari Hugging Face yang menggunakan SentencePiece tokenizer, sehingga memudahkan implementasi dan eksperimen dengan berbagai model bahasa alami.

```
!pip install zstandard
```

```
from datasets import load_dataset

# This takes a few minutes to run, so go grab a tea or coffee while you
wait :)
data_files =
"https://huggingface.co/datasets/qualis2006/PUBMED_title_abstracts_2020_ba
seline/resolve/main/PUBMED_title_abstracts_2020_baseline.jsonl.zst"
pubmed_dataset = load_dataset("json", data_files=data_files,
split="train")
pubmed_dataset
```

```
Dataset({
    features: ['meta', 'text'],
    num_rows: 15518009
})
```

```
pubmed_dataset[0]
```

```
{'meta': {'pmid': 11409574, 'language': 'eng'},
 'text': 'Epidemiology of hypoxaemia in children with acute lower respiratory
infection.\nTo determine the prevalence of hypoxaemia in children aged under 5
years suffering acute lower respiratory infections (ALRI), the risk factors for
hypoxaemia in children under 5 years of age with ALRI, and the association of
hypoxaemia with an increased risk of dying in children of the same age ...'}
```

```
!pip install psutil
```

```
import psutil

# Process.memory_info is expressed in bytes, so convert to megabytes
print(f"RAM used: {psutil.Process().memory_info().rss / (1024 * 1024):.2f}
MB")
```

```
RAM used: 5678.33 MB
```

```
print(f"Number of files in dataset : {pubmed_dataset.dataset_size}")
size_gb = pubmed_dataset.dataset_size / (1024**3)
print(f"Dataset size (cache file) : {size_gb:.2f} GB")
```

```
Number of files in dataset : 20979437051
Dataset size (cache file) : 19.54 GB
```

```
import timeit

code_snippet = """batch_size = 1000

for idx in range(0, len(pubmed_dataset), batch_size):
    _ = pubmed_dataset[idx:idx + batch_size]
"""

time = timeit.timeit(stmt=code_snippet, number=1, globals=globals())
print(
    f"Iterated over {len(pubmed_dataset)} examples (about {size_gb:.1f}
GB) in "
    f"{time:.1f}s, i.e. {size_gb/time:.3f} GB/s"
)
```

```
'Iterated over 15518009 examples (about 19.5 GB) in 64.2s, i.e. 0.304 GB/s'
```

```
next(iter(pubmed_dataset_streamed))
```

```
{'meta': {'pmid': 11409574, 'language': 'eng'},
 'text': 'Epidemiology of hypoxaemia in children with acute lower respiratory
infection.\nTo determine the prevalence of hypoxaemia in children aged under 5
years suffering acute lower respiratory infections (ALRI), the risk factors for
hypoxaemia in children under 5 years of age with ALRI, and the association of
hypoxaemia with an increased risk of dying in children of the same age ...'}
```

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
tokenized_dataset = pubmed_dataset_streamed.map(lambda x:
tokenizer(x["text"]))
next(iter(tokenized_dataset))
```

```
{'input_ids': [101, 4958, 5178, 4328, 6779, ...], 'attention_mask': [1, 1, 1, 1, 1,
...]}
```

```
shuffled_dataset = pubmed_dataset_streamed.shuffle(buffer_size=10_000,
seed=42)
next(iter(shuffled_dataset))
```

```
{'meta': {'pmid': 11410799, 'language': 'eng'},
 'text': 'Randomized study of dose or schedule modification of granulocyte colony-
stimulating factor in platinum-based chemotherapy for elderly patients with lung
cancer ...'}
```

```
dataset_head = pubmed_dataset_streamed.take(5)
list(dataset_head)
```

```
[{'meta': {'pmid': 11409574, 'language': 'eng'},
  'text': 'Epidemiology of hypoxaemia in children with acute lower respiratory
infection ...'},
 {'meta': {'pmid': 11409575, 'language': 'eng'},
  'text': 'Clinical signs of hypoxaemia in children with acute lower respiratory
infection: indicators of oxygen therapy ...'},
 {'meta': {'pmid': 11409576, 'language': 'eng'},
  'text': "Hypoxaemia in children with severe pneumonia in Papua New Guinea ..."},
 {'meta': {'pmid': 11409577, 'language': 'eng'},
  'text': 'Oxygen concentrators and cylinders ...'},
 {'meta': {'pmid': 11409578, 'language': 'eng'},
  'text': 'Oxygen supply in rural africa: a personal experience ...'}]
```

```
# Skip the first 1,000 examples and include the rest in the training set
train_dataset = shuffled_dataset.skip(1000)
# Take the first 1,000 examples for the validation set
validation_dataset = shuffled_dataset.take(1000)
```

```
law_dataset_streamed = load_dataset(
    "json",
    data_files="https://the-
eye.eu/public/AI/pile_preliminary_components/FreeLaw_Opinions.jsonl.zst",
    split="train",
    streaming=True,
)
next(iter(law_dataset_streamed))
```

```
{'meta': {'case_ID': '110921.json',
  'case_jurisdiction': 'scotus.tar.gz',
  'date_created': '2010-04-28T17:12:49Z'},
 'text': '\n461 U.S. 238 (1983)\nOLIM ET AL.\nv.\nWAKINEKONA\nNo. 81-1581.\nSupreme
Court of United States.\nArgued January 19, 1983.\nDecided April 26,
1983.\nCERTIORARI TO THE UNITED STATES COURT OF APPEALS FOR THE NINTH CIRCUIT\n*239
Michael A. Lilly, First Deputy Attorney General of Hawaii, argued the cause for
petitioners. With him on the brief was James H. Dannenberg, Deputy Attorney
General...'}
```

```
from itertools import islice
from datasets import interleave_datasets

combined_dataset = interleave_datasets([pubmed_dataset_streamed,
law_dataset_streamed])
list(islice(combined_dataset, 2))
```

```
[{'meta': {'pmid': 11409574, 'language': 'eng'},
  'text': 'Epidemiology of hypoxaemia in children with acute lower respiratory
infection ...'},
 {'meta': {'case_ID': '110921.json',
   'case_jurisdiction': 'scotus.tar.gz',
   'date_created': '2010-04-28T17:12:49Z'},
  'text': '\n461 U.S. 238 (1983)\nOLIM ET AL.\nv.\nWAKINEKONA\nNo. 81-
1581.\nSupreme Court of United States.\nArgued January 19, 1983.\nDecided April 26,
1983.\nCERTIORARI TO THE UNITED STATES COURT OF APPEALS FOR THE NINTH CIRCUIT\n*239
Michael A. Lilly, First Deputy Attorney General of Hawaii, argued the cause for
petitioners. With him on the brief was James H. Dannenberg, Deputy Attorney
General...'}]
```

```
base_url = "https://the-eye.eu/public/AI/pile/"
data_files = {
    "train": [base_url + "train/" + f"{idx:02d}.jsonl.zst" for idx in
range(30)],
    "validation": base_url + "val.jsonl.zst",
    "test": base_url + "test.jsonl.zst",
}
pile_dataset = load_dataset("json", data_files=data_files, streaming=True)
next(iter(pile_dataset["train"]))
```

```
{'meta': {'pile_set_name': 'Pile-CC'},
 'text': 'It is done, and submitted. You can play “Survival of the Tastiest” on
Android, and on the web...'}
```

Kesimpulan

Notebook ini berfungsi sebagai panduan untuk memanfaatkan pustaka seperti datasets, evaluate, dan transformers[sentencepiece] dalam menangani proyek NLP dengan dataset besar. Notebook ini dimulai dengan menginstal pustaka-pustaka tersebut untuk memungkinkan pengguna memuat, mengelola, dan mengevaluasi dataset secara efisien. Fokusnya adalah memberikan solusi praktis untuk mengolah data berskala besar dan menggunakan model NLP canggih berbasis transformer. Dengan panduan ini, pengguna dipersiapkan untuk menangani tantangan pengelolaan data dan penerapan model secara optimal, menjadikan notebook ini sebagai alat yang bermanfaat untuk eksplorasi dan pengembangan proyek berbasis data besar.

- **Creating your own dataset**

```
!pip install datasets evaluate transformers[sentencepiece]
```

⇨

Kode ini menginstal library yang diperlukan untuk pengembangan proyek NLP, termasuk datasets untuk memuat dan mengelola dataset, evaluate untuk mengevaluasi kinerja model, dan transformers[sentencepiece] untuk bekerja dengan model-model canggih dari Hugging Face yang menggunakan SentencePiece tokenizer, sehingga memudahkan implementasi dan eksperimen dengan berbagai model bahasa alami.

```
!apt install git-lfs
```

⇨

Kode ini menginstal Git LFS (Git Large File Storage) pada sistem. Git LFS digunakan untuk menangani file besar seperti model machine learning dan dataset yang tidak dapat disimpan secara efisien dalam repositori Git biasa. Dengan Git LFS, file besar dapat dikelola dan diunduh dengan lebih mudah, khususnya ketika bekerja dengan platform seperti Hugging Face Hub.

```
!git config --global user.email "m.rizkyf400@gmail.com"
!git config --global user.name "Rizkyy3"
```

⇨

Kode ini digunakan untuk mengatur konfigurasi global Git dengan menetapkan email dan nama pengguna. Konfigurasi ini memastikan setiap kontribusi pada repositori Git memiliki identitas yang jelas, memudahkan pelacakan perubahan, dan menjaga keterhubungan dengan platform pengelolaan versi seperti GitHub atau GitLab.

```
from huggingface_hub import notebook_login

notebook_login()
```

⇨

```
Kode ini digunakan untuk autentikasi ke Hugging Face Hub melalui notebook.
Dengan menjalankan notebook_login(), pengguna dapat memasukkan token akses
mereka untuk mendapatkan izin mengelola model, dataset, dan sumber daya
lain di Hugging Face Hub secara langsung dari lingkungan notebook,
sehingga mempermudah integrasi dan kolaborasi dalam proyek NLP.
```

```
!pip install requests
```

```
import requests

url =
"https://api.github.com/repos/huggingface/datasets/issues?page=1&per_page=
1"
response = requests.get(url)
```

```
response.status_code
```

```
200
```

```
response.json()
```

```
[{'url': 'https://api.github.com/repos/huggingface/datasets/issues/2792',
  'repository_url': 'https://api.github.com/repos/huggingface/datasets',
  'labels_url': 'https://api.github.com/repos/huggingface/datasets/issues/2792/labels{/name}',
  'comments_url': 'https://api.github.com/repos/huggingface/datasets/issues/2792/comments',
  'events_url': 'https://api.github.com/repos/huggingface/datasets/issues/2792/events',
  'html_url': 'https://github.com/huggingface/datasets/pull/2792',
  'id': 968650274,
  'node_id': 'MDExOlB1bGxSZXF1ZXN0NzEwNzUyMjc0',
  'number': 2792,
  'title': 'Update GooAQ',
  'user': {'login': 'bhavitvyamalik',
   'id': 19718818,
   'node_id': 'MDQ6VXNlcjE5NzE4ODE4',
   'avatar_url': 'https://avatars.githubusercontent.com/u/19718818?v=4',
   'gravatar_id': '',
   'url': 'https://api.github.com/users/bhavitvyamalik',
   'html_url': 'https://github.com/bhavitvyamalik',
   'followers_url': 'https://api.github.com/users/bhavitvyamalik/followers',
   'following_url': 'https://api.github.com/users/bhavitvyamalik/following{/other_user}',
   'gists_url': 'https://api.github.com/users/bhavitvyamalik/gists{/gist_id}',
   'starred_url': 'https://api.github.com/users/bhavitvyamalik/starred{/owner}{/repo}',
   'subscriptions_url': 'https://api.github.com/users/bhavitvyamalik/subscriptions',
   'organizations_url': 'https://api.github.com/users/bhavitvyamalik/orgs',
   'repos_url': 'https://api.github.com/users/bhavitvyamalik/repos',
   'events_url': 'https://api.github.com/users/bhavitvyamalik/events{/privacy}',
   'received_events_url': 'https://api.github.com/users/bhavitvyamalik/received_events',
   'type': 'User',
   'site_admin': False},
  'labels': [],
  'state': 'open',
  'locked': False,
  'assignee': None,
  'assignees': [],
  'milestone': None,
  'comments': 1,
  'created_at': '2021-08-12T11:40:18Z',
  'updated_at': '2021-08-12T12:31:17Z',
  'closed_at': None,
  'author_association': 'CONTRIBUTOR',
  'active_lock_reason': None,
  'pull_request': {'url': 'https://api.github.com/repos/huggingface/datasets/pulls/2792',
   'html_url': 'https://github.com/huggingface/datasets/pull/2792',
   'diff_url': 'https://github.com/huggingface/datasets/pull/2792.diff',
   'patch_url': 'https://github.com/huggingface/datasets/pull/2792.patch'},
  'body': '[GooAQ](https://github.com/allenai/gooaq) dataset was recently updated after splits
were added for the same. This PR contains new updated GooAQ with train/val/test splits and updated
README as well.',
  'performed_via_github_app': None}]
```

```
GITHUB_TOKEN = xxx  # Copy your GitHub token here
headers = {"Authorization": f"token {GITHUB_TOKEN}"}
```

```
import time
import math
from pathlib import Path
import pandas as pd
from tqdm.notebook import tqdm


def fetch_issues(
```

```
    owner="huggingface",
    repo="datasets",
    num_issues=10_000,
    rate_limit=5_000,
    issues_path=Path("."),
):
    if not issues_path.is_dir():
        issues_path.mkdir(exist_ok=True)

    batch = []
    all_issues = []
    per_page = 100  # Number of issues to return per page
    num_pages = math.ceil(num_issues / per_page)
    base_url = "https://api.github.com/repos"

    for page in tqdm(range(num_pages)):
        # Query with state=all to get both open and closed issues
        query = f"issues?page={page}&per_page={per_page}&state=all"
        issues = requests.get(f"{base_url}/{owner}/{repo}/{query}",
headers=headers)
        batch.extend(issues.json())

        if len(batch) > rate_limit and len(all_issues) < num_issues:
            all_issues.extend(batch)
            batch = []  # Flush batch for next time period
            print(f"Reached GitHub rate limit. Sleeping for one hour ...")
            time.sleep(60 * 60 + 1)

    all_issues.extend(batch)
    df = pd.DataFrame.from_records(all_issues)
    df.to_json(f"{issues_path}/{repo}-issues.jsonl", orient="records",
lines=True)
    print(
        f"Downloaded all the issues for {repo}! Dataset stored at
{issues_path}/{repo}-issues.jsonl"
    )

# Depending on your internet connection, this can take several minutes to
run...
fetch_issues()
```

```
issues_dataset = load_dataset("json", data_files="datasets-issues.jsonl",
split="train")
issues_dataset
```

```
Dataset({
    features: ['url', 'repository_url', 'labels_url', 'comments_url', 'events_url', 'html_url',
'id', 'node_id', 'number', 'title', 'user', 'labels', 'state', 'locked', 'assignee', 'assignees',
'milestone', 'comments', 'created_at', 'updated_at', 'closed_at', 'author_association',
'active_lock_reason', 'pull_request', 'body', 'timeline_url', 'performed_via_github_app'],
    num_rows: 3019
})
```

```
sample = issues_dataset.shuffle(seed=666).select(range(3))

# Print out the URL and pull request entries
for url, pr in zip(sample["html_url"], sample["pull_request"]):
    print(f">> URL: {url}")
    print(f">> Pull request: {pr}\n")
```

```
>> URL: https://github.com/huggingface/datasets/pull/850
>> Pull request: {'url': 'https://api.github.com/repos/huggingface/datasets/pulls/850',
'html_url': 'https://github.com/huggingface/datasets/pull/850', 'diff_url':
'https://github.com/huggingface/datasets/pull/850.diff', 'patch_url':
'https://github.com/huggingface/datasets/pull/850.patch'}

>> URL: https://github.com/huggingface/datasets/issues/2773
>> Pull request: None

>> URL: https://github.com/huggingface/datasets/pull/783
>> Pull request: {'url': 'https://api.github.com/repos/huggingface/datasets/pulls/783',
'html_url': 'https://github.com/huggingface/datasets/pull/783', 'diff_url':
'https://github.com/huggingface/datasets/pull/783.diff', 'patch_url':
'https://github.com/huggingface/datasets/pull/783.patch'}
```

```
issues_dataset = issues_dataset.map(
    lambda x: {"is_pull_request": False if x["pull_request"] is None else
True}
)
```

```
issue_number = 2792
url =
f"https://api.github.com/repos/huggingface/datasets/issues/{issue_number}/
comments"
response = requests.get(url, headers=headers)
response.json()
```

```
[{'url': 'https://api.github.com/repos/huggingface/datasets/issues/comments/897594128',
  'html_url': 'https://github.com/huggingface/datasets/pull/2792#issuecomment-897594128',
  'issue_url': 'https://api.github.com/repos/huggingface/datasets/issues/2792',
  'id': 897594128,
  'node_id': 'IC_kwDODunzps41gDMQ',
  'user': {'login': 'bhavitvyamalik',
   'id': 19718818,
   'node_id': 'MDQ6VXNlcjE5NzE4ODE4',
   'avatar_url': 'https://avatars.githubusercontent.com/u/19718818?v=4',
   'gravatar_id': '',
   'url': 'https://api.github.com/users/bhavitvyamalik',
   'html_url': 'https://github.com/bhavitvyamalik',
   'followers_url': 'https://api.github.com/users/bhavitvyamalik/followers',
   'following_url': 'https://api.github.com/users/bhavitvyamalik/following{/other_user}',
   'gists_url': 'https://api.github.com/users/bhavitvyamalik/gists{/gist_id}',
   'starred_url': 'https://api.github.com/users/bhavitvyamalik/starred{/owner}{/repo}',
   'subscriptions_url': 'https://api.github.com/users/bhavitvyamalik/subscriptions',
   'organizations_url': 'https://api.github.com/users/bhavitvyamalik/orgs',
   'repos_url': 'https://api.github.com/users/bhavitvyamalik/repos',
   'events_url': 'https://api.github.com/users/bhavitvyamalik/events{/privacy}',
   'received_events_url': 'https://api.github.com/users/bhavitvyamalik/received_events',
   'type': 'User',
   'site_admin': False},
  'created_at': '2021-08-12T12:21:52Z',
  'updated_at': '2021-08-12T12:31:17Z',
  'author_association': 'CONTRIBUTOR',
  'body': "@albertvillanova my tests are failing here:\r\n```\r\ndataset_name = 'gooaq'\r\n\r\n
def test_load_dataset(self, dataset_name):\r\n        configs =
self.dataset_tester.load_all_configs(dataset_name, is_local=True)[:1]\r\n>
self.dataset_tester.check_load_dataset(dataset_name, configs, is_local=True,
use_local_dummy_data=True)\r\n\r\ntests/test_dataset_common.py:234: \r\n_ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ \r\ntests/test_dataset_common.py:187: in
check_load_dataset\r\n    self.parent.assertTrue(len(dataset[split]) > 0)\r\nE   AssertionError:
False is not true\r\n```\r\nWhen I try loading dataset on local machine it works fine. Any
suggestions on how can I avoid this error?",
  'performed_via_github_app': None}]
```

```
def get_comments(issue_number):
    url =
f"https://api.github.com/repos/huggingface/datasets/issues/{issue_number}/
comments"
    response = requests.get(url, headers=headers)
    return [r["body"] for r in response.json()]


# Test our function works as expected
get_comments(2792)
```

```
["@albertvillanova my tests are failing here:\r\n```\r\ndataset_name = 'gooaq'\r\n\r\n    def
test_load_dataset(self, dataset_name):\r\n        configs =
self.dataset_tester.load_all_configs(dataset_name, is_local=True)[:1]\r\n>
self.dataset_tester.check_load_dataset(dataset_name, configs, is_local=True,
use_local_dummy_data=True)\r\n\r\ntests/test_dataset_common.py:234: \r\n_ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ \r\ntests/test_dataset_common.py:187: in
check_load_dataset\r\n    self.parent.assertTrue(len(dataset[split]) > 0)\r\nE   AssertionError:
False is not true\r\n```\r\nWhen I try loading dataset on local machine it works fine. Any
suggestions on how can I avoid this error?"]
```

```
# Depending on your internet connection, this can take a few minutes...
issues_with_comments_dataset = issues_dataset.map(
    lambda x: {"comments": get_comments(x["number"])})
```

```
from huggingface_hub import notebook_login

notebook_login()
```

```
issues_with_comments_dataset.push_to_hub("github-issues")
```

```
remote_dataset = load_dataset("lewtun/github-issues", split="train")
remote_dataset
```

```
Dataset({
    features: ['url', 'repository_url', 'labels_url', 'comments_url', 'events_url', 'html_url',
'id', 'node_id', 'number', 'title', 'user', 'labels', 'state', 'locked', 'assignee', 'assignees',
'milestone', 'comments', 'created_at', 'updated_at', 'closed_at', 'author_association',
'active_lock_reason', 'pull_request', 'body', 'performed_via_github_app', 'is_pull_request'],
    num_rows: 2855
})
```

Kesimpulan

Notebook ini bertujuan untuk membuat dan mengelola dataset kustom menggunakan pustaka Hugging Face, dimulai dengan instalasi dan konfigurasi alat seperti datasets, evaluate, dan transformers, serta login ke Hugging Face Hub. Selanjutnya, notebook mencakup pengambilan data melalui API GitHub menggunakan autentikasi token, diikuti dengan pemrosesan dataset untuk keperluan analisis dan pelatihan model. Selain itu, terdapat langkah-langkah evaluasi dataset dan penerapan model transformator untuk mendukung pengolahan data berbasis machine learning secara efisien.

- **Semantic search with FAISS**

```
!pip install datasets evaluate transformers[sentencepiece]
```

```
!pip install faiss-gpu
```

```
from datasets import load_dataset

issues_dataset = load_dataset("lewtun/github-issues", split="train")
issues_dataset
```

```
Dataset({
    features: ['url', 'repository_url', 'labels_url', 'comments_url', 'events_url', 'html_url',
'id', 'node_id', 'number', 'title', 'user', 'labels', 'state', 'locked', 'assignee', 'assignees',
'milestone', 'comments', 'created_at', 'updated_at', 'closed_at', 'author_association',
'active_lock_reason', 'pull_request', 'body', 'performed_via_github_app', 'is_pull_request'],
    num_rows: 2855
})
```

```
issues_dataset = issues_dataset.filter(
    lambda x: (x["is_pull_request"] == False and len(x["comments"]) > 0)
)
issues_dataset
```

```
Dataset({
    features: ['url', 'repository_url', 'labels_url', 'comments_url', 'events_url', 'html_url',
'id', 'node_id', 'number', 'title', 'user', 'labels', 'state', 'locked', 'assignee', 'assignees',
'milestone', 'comments', 'created_at', 'updated_at', 'closed_at', 'author_association',
'active_lock_reason', 'pull_request', 'body', 'performed_via_github_app', 'is_pull_request'],
    num_rows: 771
})
```

```
columns = issues_dataset.column_names
columns_to_keep = ["title", "body", "html_url", "comments"]
columns_to_remove = set(columns_to_keep).symmetric_difference(columns)
issues_dataset = issues_dataset.remove_columns(columns_to_remove)
issues_dataset
```

```
Dataset({
    features: ['html_url', 'title', 'comments', 'body'],
    num_rows: 771
})
```

```
issues_dataset.set_format("pandas")
df = issues_dataset[:]
```

```
df["comments"][0].tolist()
```

```
['the bug code locate in : \r\n    if data_args.task_name is not None:\r\n        # Downloading
and loading a dataset from the hub.\r\n        datasets = load_dataset("glue",
data_args.task_name, cache_dir=model_args.cache_dir)',
 'Hi @jinec,\r\n\r\nFrom time to time we get this kind of `ConnectionError` coming from the
github.com website: https://raw.githubusercontent.com\r\n\r\nNormally, it should work if you wait
a little and then retry.\r\n\r\nCould you please confirm if the problem persists?',
 'cannot connect, even by Web browser, please check that  there is some  problems。',
 'I can access https://raw.githubusercontent.com/huggingface/datasets/1.7.0/datasets/glue/glue.py
without problem...']
```

```
comments_df = df.explode("comments", ignore_index=True)
comments_df.head(4)
```

```
from datasets import Dataset

comments_dataset = Dataset.from_pandas(comments_df)
comments_dataset
```

```
Dataset({
    features: ['html_url', 'title', 'comments', 'body'],
    num_rows: 2842
})
```

```
comments_dataset = comments_dataset.map(
    lambda x: {"comment_length": len(x["comments"].split())}
)
```

```
comments_dataset = comments_dataset.filter(lambda x: x["comment_length"] > 15)
comments_dataset
```

```
Dataset({
    features: ['html_url', 'title', 'comments', 'body', 'comment_length'],
    num_rows: 2098
})
```

```
def concatenate_text(examples):
    return {
        "text": examples["title"]
        + " \n "
        + examples["body"]
        + " \n "
        + examples["comments"]
    }


comments_dataset = comments_dataset.map(concatenate_text)
```

```
from transformers import AutoTokenizer, AutoModel

model_ckpt = "sentence-transformers/multi-qa-mpnet-base-dot-v1"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
model = AutoModel.from_pretrained(model_ckpt)
```

```
import torch

device = torch.device("cuda")
model.to(device)
```

```
def cls_pooling(model_output):
    return model_output.last_hidden_state[:, 0]
```

```
def get_embeddings(text_list):
    encoded_input = tokenizer(
        text_list, padding=True, truncation=True, return_tensors="pt"
    )
    encoded_input = {k: v.to(device) for k, v in encoded_input.items()}
    model_output = model(**encoded_input)
    return cls_pooling(model_output)
```

```
embedding = get_embeddings(comments_dataset["text"][0])
embedding.shape
```

```
torch.Size([1, 768])
```

```
embeddings_dataset = comments_dataset.map(
    lambda x: {"embeddings":
get_embeddings(x["text"]).detach().cpu().numpy()[0]}
)
```

```
embeddings_dataset.add_faiss_index(column="embeddings")
```

```
question = "How can I load a dataset offline?"
question_embedding = get_embeddings([question]).cpu().detach().numpy()
question_embedding.shape
```

```
torch.Size([1, 768])
```

```
scores, samples = embeddings_dataset.get_nearest_examples(
    "embeddings", question_embedding, k=5
)
```

```
import pandas as pd

samples_df = pd.DataFrame.from_dict(samples)
samples_df["scores"] = scores
samples_df.sort_values("scores", ascending=False, inplace=True)
```

```
for _, row in samples_df.iterrows():
    print(f"COMMENT: {row.comments}")
    print(f"SCORE: {row.scores}")
    print(f"TITLE: {row.title}")
    print(f"URL: {row.html_url}")
    print("=" * 50)
    print()
```

```
"""
COMMENT: Requiring online connection is a deal breaker in some cases unfortunately so it'd be
great if offline mode is added similar to how `transformers` loads models offline fine.

@mandubian's second bullet point suggests that there's a workaround allowing you to use your
offline (custom?) dataset with `datasets`. Could you please elaborate on how that should look
like?
SCORE: 25.505046844482422
TITLE: Discussion using datasets in offline mode
URL: https://github.com/huggingface/datasets/issues/824
==================================================

COMMENT: The local dataset builders (csv, text , json and pandas) are now part of the `datasets`
package since #1726 :)
You can now use them offline
\`\`\`python
datasets = load_dataset("text", data_files=data_files)
\`\`\`

We'll do a new release soon
SCORE: 24.555509567260742
TITLE: Discussion using datasets in offline mode
URL: https://github.com/huggingface/datasets/issues/824
==================================================

COMMENT: I opened a PR that allows to reload modules that have already been loaded once even if
there's no internet.

Let me know if you know other ways that can make the offline mode experience better. I'd be happy
to add them :)

I already note the "freeze" modules option, to prevent local modules updates. It would be a cool
feature.

----------
```

6. The Tokenizers Library

- **Training a new tokenizer from an old one**

```
!pip install datasets evaluate transformers[sentencepiece]
```

```
!apt install git-lfs
```

```
!git config --global user.email "m.rizkyf400@gmail.com"
!git config --global user.name "Rizkyy3"
```

```
from huggingface_hub import notebook_login

notebook_login()
```

```python
from datasets import load_dataset

# This can take a few minutes to load, so grab a coffee or tea while you wait!
raw_datasets = load_dataset("code_search_net", "python")
```

```python
raw_datasets["train"]
```

```
Dataset({
    features: ['repository_name', 'func_path_in_repository', 'func_name', 'whole_func_string', 'language',
     'func_code_string', 'func_code_tokens', 'func_documentation_string', 'func_documentation_tokens', 'split_name',
     'func_code_url'
    ],
    num_rows: 412178
})
```

```python
print(raw_datasets["train"][123456]["whole_func_string"])
```

```python
# Don't uncomment the following line unless your dataset is small!
# training_corpus = [raw_datasets["train"][i: i + 1000]["whole_func_string"] for i in range(0, len(raw_datasets["train"]), 1000)]
```

```python
training_corpus = (
    raw_datasets["train"][i : i + 1000]["whole_func_string"]
    for i in range(0, len(raw_datasets["train"]), 1000)
)
```

```python
gen = (i for i in range(10))
print(list(gen))
print(list(gen))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[]
```

```python
def get_training_corpus():
    return (
        raw_datasets["train"][i : i + 1000]["whole_func_string"]
        for i in range(0, len(raw_datasets["train"]), 1000)
    )


training_corpus = get_training_corpus()
```

```
def get_training_corpus():
    dataset = raw_datasets["train"]
    for start_idx in range(0, len(dataset), 1000):
        samples = dataset[start_idx : start_idx + 1000]
        yield samples["whole_func_string"]
```

```
from transformers import AutoTokenizer

old_tokenizer = AutoTokenizer.from_pretrained("gpt2")
```

```
example = '''def add_numbers(a, b):
    """Add the two numbers `a` and `b`."""
    return a + b'''

tokens = old_tokenizer.tokenize(example)
tokens
```

```
['def', 'Ġadd', '_', 'n', 'umbers', '(', 'a', ',', 'Ġb', '):', 'Ċ', 'Ġ', 'Ġ', 'Ġ', 'Ġ"""', 'Add',
'Ġthe', 'Ġtwo',
 'Ġnumbers', 'Ġ`', 'a', '`', 'Ġand', 'Ġ`', 'b', '`', '."', '"""', 'Ċ', 'Ġ', 'Ġ', 'Ġ', 'Ġreturn',
'Ġa', 'Ġ+', 'Ġb']
```

```
tokenizer = old_tokenizer.train_new_from_iterator(training_corpus, 52000)
```

```
tokens = tokenizer.tokenize(example)
tokens
```

```
['def', 'Ġadd', '_', 'numbers', '(', 'a', ',', 'Ġb', '):', 'ĊĠĠĠ', 'Ġ"""', 'Add', 'Ġthe', 'Ġtwo',
'Ġnumbers', 'Ġ`',
 'a', '`', 'Ġand', 'Ġ`', 'b', '`."""', 'ĊĠĠĠ', 'Ġreturn', 'Ġa', 'Ġ+', 'Ġb']
```

```
print(len(tokens))
print(len(old_tokenizer.tokenize(example)))
```

```
27
36
```

```
example = """class LinearLayer():
    def __init__(self, input_size, output_size):
        self.weight = torch.randn(input_size, output_size)
        self.bias = torch.zeros(output_size)

    def __call__(self, x):
        return x @ self.weights + self.bias
    """
tokenizer.tokenize(example)
```

```
['class', 'ĠLinear', 'Layer', '():', 'ĊĠĠĠ', 'Ġdef', 'Ġ__', 'init', '__(', 'self', ',', 'Ġinput',
'_', 'size', ',',
 'Ġoutput', '_', 'size', '):', 'ĊĠĠĠĠĠĠĠ', 'Ġself', '.', 'weight', 'Ġ=', 'Ġtorch', '.', 'randn',
'(', 'input', '_',
 'size', ',', 'Ġoutput', '_', 'size', ')', 'ĊĠĠĠĠĠĠĠ', 'Ġself', '.', 'bias', 'Ġ=', 'Ġtorch', '.',
'zeros', '(',
 'output', '_', 'size', ')', 'ĊĊĠĠĠ', 'Ġdef', 'Ġ__', 'call', '__(', 'self', ',', 'Ġx', '):',
'ĊĠĠĠĠĠĠĠ',
 'Ġreturn', 'Ġx', 'Ġ@', 'Ġself', '.', 'weights', 'Ġ+', 'Ġself', '.', 'bias', 'ĊĠĠĠ']
```

```
tokenizer.save_pretrained("code-search-net-tokenizer")
```

```
from huggingface_hub import notebook_login

notebook_login()
```

```
tokenizer.push_to_hub("code-search-net-tokenizer")
```

```
# Replace "huggingface-course" below with your actual namespace to use
your own tokenizer
tokenizer = AutoTokenizer.from_pretrained("huggingface-course/code-search-
net-tokenizer")
```

- **Fast tokenizers' special powers**

```
!pip install datasets evaluate transformers[sentencepiece]
```

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
example = "My name is Sylvain and I work at Hugging Face in Brooklyn."
encoding = tokenizer(example)
print(type(encoding))
```

```
<class 'transformers.tokenization_utils_base.BatchEncoding'>
```

```
tokenizer.is_fast
```

```
True
```

```
encoding.is_fast
```

```
True
```

```
encoding.tokens()
```

```
['[CLS]', 'My', 'name', 'is', 'S', '##yl', '##va', '##in', 'and', 'I', 'work', 'at', 'Hu',
'##gging', 'Face', 'in',
 'Brooklyn', '.', '[SEP]']
```

```
encoding.word_ids()
```

```
[None, 0, 1, 2, 3, 3, 3, 3, 4, 5, 6, 7, 8, 8, 9, 10, 11, 12, None]
```

```
start, end = encoding.word_to_chars(3)
example[start:end]
```

```
Sylvain
```

```
from transformers import pipeline

token_classifier = pipeline("token-classification")
token_classifier("My name is Sylvain and I work at Hugging Face in
Brooklyn.")
```

```
[{'entity': 'I-PER', 'score': 0.9993828, 'index': 4, 'word': 'S', 'start': 11, 'end': 12},
 {'entity': 'I-PER', 'score': 0.99815476, 'index': 5, 'word': '##yl', 'start': 12, 'end': 14},
 {'entity': 'I-PER', 'score': 0.99590725, 'index': 6, 'word': '##va', 'start': 14, 'end': 16},
 {'entity': 'I-PER', 'score': 0.9992327, 'index': 7, 'word': '##in', 'start': 16, 'end': 18},
 {'entity': 'I-ORG', 'score': 0.97389334, 'index': 12, 'word': 'Hu', 'start': 33, 'end': 35},
 {'entity': 'I-ORG', 'score': 0.976115, 'index': 13, 'word': '##gging', 'start': 35, 'end': 40},
 {'entity': 'I-ORG', 'score': 0.98879766, 'index': 14, 'word': 'Face', 'start': 41, 'end': 45},
 {'entity': 'I-LOC', 'score': 0.99321055, 'index': 16, 'word': 'Brooklyn', 'start': 49, 'end':
57}]
```

```
from transformers import pipeline

token_classifier = pipeline("token-classification",
aggregation_strategy="simple")
token_classifier("My name is Sylvain and I work at Hugging Face in
Brooklyn.")
```

```
[{'entity_group': 'PER', 'score': 0.9981694, 'word': 'Sylvain', 'start': 11, 'end': 18},
 {'entity_group': 'ORG', 'score': 0.97960204, 'word': 'Hugging Face', 'start': 33, 'end': 45},
 {'entity_group': 'LOC', 'score': 0.99321055, 'word': 'Brooklyn', 'start': 49, 'end': 57}]
```

```python
from transformers import AutoTokenizer, AutoModelForTokenClassification

model_checkpoint = "dbmdz/bert-large-cased-finetuned-conll03-english"
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
model = AutoModelForTokenClassification.from_pretrained(model_checkpoint)

example = "My name is Sylvain and I work at Hugging Face in Brooklyn."
inputs = tokenizer(example, return_tensors="pt")
outputs = model(**inputs)
```

```python
print(inputs["input_ids"].shape)
print(outputs.logits.shape)
```

```
torch.Size([1, 19])
torch.Size([1, 19, 9])
```

```python
import torch

probabilities = torch.nn.functional.softmax(outputs.logits, dim=-
1)[0].tolist()
predictions = outputs.logits.argmax(dim=-1)[0].tolist()
print(predictions)
```

```
[0, 0, 0, 0, 4, 4, 4, 4, 0, 0, 0, 0, 6, 6, 6, 0, 8, 0, 0]
```

```python
model.config.id2label
```

```
{0: 'O',
 1: 'B-MISC',
 2: 'I-MISC',
 3: 'B-PER',
 4: 'I-PER',
 5: 'B-ORG',
 6: 'I-ORG',
 7: 'B-LOC',
 8: 'I-LOC'}
```

```
results = []
tokens = inputs.tokens()

for idx, pred in enumerate(predictions):
    label = model.config.id2label[pred]
    if label != "O":
        results.append(
            {"entity": label, "score": probabilities[idx][pred], "word":
tokens[idx]}
        )

print(results)
```

```
[{'entity': 'I-PER', 'score': 0.9993828, 'index': 4, 'word': 'S'},
 {'entity': 'I-PER', 'score': 0.99815476, 'index': 5, 'word': '##yl'},
 {'entity': 'I-PER', 'score': 0.99590725, 'index': 6, 'word': '##va'},
 {'entity': 'I-PER', 'score': 0.9992327, 'index': 7, 'word': '##in'},
 {'entity': 'I-ORG', 'score': 0.97389334, 'index': 12, 'word': 'Hu'},
 {'entity': 'I-ORG', 'score': 0.976115, 'index': 13, 'word': '##gging'},
 {'entity': 'I-ORG', 'score': 0.98879766, 'index': 14, 'word': 'Face'},
 {'entity': 'I-LOC', 'score': 0.99321055, 'index': 16, 'word': 'Brooklyn'}]
```

```
inputs_with_offsets = tokenizer(example, return_offsets_mapping=True)
inputs_with_offsets["offset_mapping"]
```

```
[(0, 0), (0, 2), (3, 7), (8, 10), (11, 12), (12, 14), (14, 16), (16, 18), (19, 22), (23, 24), (25,
29), (30, 32),
 (33, 35), (35, 40), (41, 45), (46, 48), (49, 57), (57, 58), (0, 0)]
```

```
example[12:14]
```

```
yl
```

```
results = []
inputs_with_offsets = tokenizer(example, return_offsets_mapping=True)
tokens = inputs_with_offsets.tokens()
offsets = inputs_with_offsets["offset_mapping"]

for idx, pred in enumerate(predictions):
    label = model.config.id2label[pred]
    if label != "O":
        start, end = offsets[idx]
        results.append(
            {
```

```
                "entity": label,
                "score": probabilities[idx][pred],
                "word": tokens[idx],
                "start": start,
                "end": end,
            }
        )

print(results)
```

```
[{'entity': 'I-PER', 'score': 0.9993828, 'index': 4, 'word': 'S', 'start': 11, 'end': 12},
 {'entity': 'I-PER', 'score': 0.99815476, 'index': 5, 'word': '##yl', 'start': 12, 'end': 14},
 {'entity': 'I-PER', 'score': 0.99590725, 'index': 6, 'word': '##va', 'start': 14, 'end': 16},
 {'entity': 'I-PER', 'score': 0.9992327, 'index': 7, 'word': '##in', 'start': 16, 'end': 18},
 {'entity': 'I-ORG', 'score': 0.97389334, 'index': 12, 'word': 'Hu', 'start': 33, 'end': 35},
 {'entity': 'I-ORG', 'score': 0.976115, 'index': 13, 'word': '##gging', 'start': 35, 'end': 40},
 {'entity': 'I-ORG', 'score': 0.98879766, 'index': 14, 'word': 'Face', 'start': 41, 'end': 45},
 {'entity': 'I-LOC', 'score': 0.99321055, 'index': 16, 'word': 'Brooklyn', 'start': 49, 'end':
57}]
```

```
example[33:45]
```

```
Hugging Face
```

```
import numpy as np

results = []
inputs_with_offsets = tokenizer(example, return_offsets_mapping=True)
tokens = inputs_with_offsets.tokens()
offsets = inputs_with_offsets["offset_mapping"]

idx = 0
while idx < len(predictions):
    pred = predictions[idx]
    label = model.config.id2label[pred]
    if label != "O":
        # Remove the B- or I-
        label = label[2:]
        start, _ = offsets[idx]

        # Grab all the tokens labeled with I-label
        all_scores = []
        while (
            idx < len(predictions)
            and model.config.id2label[predictions[idx]] == f"I-{label}"
        ):
```

```
            all_scores.append(probabilities[idx][pred])
            _, end = offsets[idx]
            idx += 1

        # The score is the mean of all the scores of the tokens in that
grouped entity
        score = np.mean(all_scores).item()
        word = example[start:end]
        results.append(
            {
                "entity_group": label,
                "score": score,
                "word": word,
                "start": start,
                "end": end,
            }
        )
    idx += 1

print(results)
```

```
[{'entity_group': 'PER', 'score': 0.9981694, 'word': 'Sylvain', 'start': 11, 'end': 18},
 {'entity_group': 'ORG', 'score': 0.97960204, 'word': 'Hugging Face', 'start': 33, 'end': 45},
 {'entity_group': 'LOC', 'score': 0.99321055, 'word': 'Brooklyn', 'start': 49, 'end': 57}]
```

- **Fast tokenizers in the QA pipeline**

```
!pip install datasets evaluate transformers[sentencepiece]
```

```
from transformers import pipeline

question_answerer = pipeline("question-answering")
context = """
🤗 Transformers is backed by the three most popular deep learning
libraries — Jax, PyTorch, and TensorFlow — with a seamless integration
between them. It's straightforward to train your models with one before
loading them for inference with the other.
"""
question = "Which deep learning libraries back 🤗 Transformers?"
question_answerer(question=question, context=context)
```

```
{'score': 0.97773,
 'start': 78,
 'end': 105,
 'answer': 'Jax, PyTorch and TensorFlow'}
```

```
long_context = """
🤗 Transformers: State of the Art NLP

🤗 Transformers provides thousands of pretrained models to perform tasks
on texts such as classification, information extraction,
question answering, summarization, translation, text generation and more
in over 100 languages.
Its aim is to make cutting-edge NLP easier to use for everyone.

🤗 Transformers provides APIs to quickly download and use those pretrained
models on a given text, fine-tune them on your own datasets and
then share them with the community on our model hub. At the same time,
each python module defining an architecture is fully standalone and
can be modified to enable quick research experiments.

Why should I use transformers?

1. Easy-to-use state-of-the-art models:
  - High performance on NLU and NLG tasks.
  - Low barrier to entry for educators and practitioners.
  - Few user-facing abstractions with just three classes to learn.
  - A unified API for using all our pretrained models.
  - Lower compute costs, smaller carbon footprint:

2. Researchers can share trained models instead of always retraining.
  - Practitioners can reduce compute time and production costs.
  - Dozens of architectures with over 10,000 pretrained models, some in
more than 100 languages.

3. Choose the right framework for every part of a model's lifetime:
  - Train state-of-the-art models in 3 lines of code.
  - Move a single model between TF2.0/PyTorch frameworks at will.
  - Seamlessly pick the right framework for training, evaluation and
production.

4. Easily customize a model or an example to your needs:
  - We provide examples for each architecture to reproduce the results
published by its original authors.
  - Model internals are exposed as consistently as possible.
  - Model files can be used independently of the library for quick
experiments.

🤗 Transformers is backed by the three most popular deep learning
libraries — Jax, PyTorch and TensorFlow — with a seamless integration
```

```
between them. It's straightforward to train your models with one before
loading them for inference with the other.
"""
question_answerer(question=question, context=long_context)
```

```
{'score': 0.97149,
 'start': 1892,
 'end': 1919,
 'answer': 'Jax, PyTorch and TensorFlow'}
```

```
from transformers import AutoTokenizer, AutoModelForQuestionAnswering

model_checkpoint = "distilbert-base-cased-distilled-squad"
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
model = AutoModelForQuestionAnswering.from_pretrained(model_checkpoint)

inputs = tokenizer(question, context, return_tensors="pt")
outputs = model(**inputs)
```

```
start_logits = outputs.start_logits
end_logits = outputs.end_logits
print(start_logits.shape, end_logits.shape)
```

```
torch.Size([1, 66]) torch.Size([1, 66])
```

```
import torch

sequence_ids = inputs.sequence_ids()
# Mask everything apart from the tokens of the context
mask = [i != 1 for i in sequence_ids]
# Unmask the [CLS] token
mask[0] = False
mask = torch.tensor(mask)[None]

start_logits[mask] = -10000
end_logits[mask] = -10000
```

```
start_probabilities = torch.nn.functional.softmax(start_logits, dim=-1)[0]
end_probabilities = torch.nn.functional.softmax(end_logits, dim=-1)[0]
```

```
scores = start_probabilities[:, None] * end_probabilities[None, :]
```

```
scores = torch.triu(scores)
```

```
max_index = scores.argmax().item()
start_index = max_index // scores.shape[1]
end_index = max_index % scores.shape[1]
print(scores[start_index, end_index])
```

```
0.97773
```

```
inputs_with_offsets = tokenizer(question, context,
return_offsets_mapping=True)
offsets = inputs_with_offsets["offset_mapping"]

start_char, _ = offsets[start_index]
_, end_char = offsets[end_index]
answer = context[start_char:end_char]
```

```
result = {
    "answer": answer,
    "start": start_char,
    "end": end_char,
    "score": scores[start_index, end_index],
}
print(result)
```

```
{'answer': 'Jax, PyTorch and TensorFlow',
 'start': 78,
 'end': 105,
 'score': 0.97773}
```

```
inputs = tokenizer(question, long_context)
print(len(inputs["input_ids"]))
```

```
461
```

```
inputs = tokenizer(question, long_context, max_length=384,
truncation="only_second")
print(tokenizer.decode(inputs["input_ids"]))
```

```
"""
[CLS] Which deep learning libraries back [UNK] Transformers? [SEP] [UNK] Transformers : State of
the Art NLP

[UNK] Transformers provides thousands of pretrained models to perform tasks on texts such as
classification, information extraction,
question answering, summarization, translation, text generation and more in over 100 languages.
Its aim is to make cutting-edge NLP easier to use for everyone.

[UNK] Transformers provides APIs to quickly download and use those pretrained models on a given
text, fine-tune them on your own datasets and
then share them with the community on our model hub. At the same time, each python module defining
an architecture is fully standalone and
can be modified to enable quick research experiments.

Why should I use transformers?

1. Easy-to-use state-of-the-art models:
  - High performance on NLU and NLG tasks.
  - Low barrier to entry for educators and practitioners.
  - Few user-facing abstractions with just three classes to learn.
  - A unified API for using all our pretrained models.
  - Lower compute costs, smaller carbon footprint:

2. Researchers can share trained models instead of always retraining.
  - Practitioners can reduce compute time and production costs.
  - Dozens of architectures with over 10,000 pretrained models, some in more than 100 languages.

3. Choose the right framework for every part of a model's lifetime:
  - Train state-of-the-art models in 3 lines of code.
  - Move a single model between TF2.0/PyTorch frameworks at will.
  - Seamlessly pick the right framework for training, evaluation and production.

4. Easily customize a model or an example to your needs:
  - We provide examples for each architecture to reproduce the results published by its original
authors.
  - Model internal [SEP]
"""
```

```
sentence = "This sentence is not too long but we are going to split it
anyway."
inputs = tokenizer(
    sentence, truncation=True, return_overflowing_tokens=True,
max_length=6, stride=2
)

for ids in inputs["input_ids"]:
    print(tokenizer.decode(ids))
```

```
'[CLS] This sentence is not [SEP]'
'[CLS] is not too long [SEP]'
'[CLS] too long but we [SEP]'
'[CLS] but we are going [SEP]'
'[CLS] are going to split [SEP]'
'[CLS] to split it anyway [SEP]'
'[CLS] it anyway. [SEP]'
```

```
print(inputs.keys())
```

```
dict_keys(['input_ids', 'attention_mask', 'overflow_to_sample_mapping'])
```

```
print(inputs["overflow_to_sample_mapping"])
```

```
[0, 0, 0, 0, 0, 0, 0]
```

```
sentences = [
    "This sentence is not too long but we are going to split it anyway.",
    "This sentence is shorter but will still get split.",
]
inputs = tokenizer(
    sentences, truncation=True, return_overflowing_tokens=True,
max_length=6, stride=2
)

print(inputs["overflow_to_sample_mapping"])
```

```
[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
```

```
inputs = tokenizer(
    question,
    long_context,
    stride=128,
    max_length=384,
    padding="longest",
    truncation="only_second",
    return_overflowing_tokens=True,
    return_offsets_mapping=True,
)
```

```
_ = inputs.pop("overflow_to_sample_mapping")
offsets = inputs.pop("offset_mapping")

inputs = inputs.convert_to_tensors("pt")
print(inputs["input_ids"].shape)
```

```
torch.Size([2, 384])
```

```
outputs = model(**inputs)

start_logits = outputs.start_logits
end_logits = outputs.end_logits
print(start_logits.shape, end_logits.shape)
```

```
torch.Size([2, 384]) torch.Size([2, 384])
```

```
sequence_ids = inputs.sequence_ids()
# Mask everything apart from the tokens of the context
mask = [i != 1 for i in sequence_ids]
# Unmask the [CLS] token
mask[0] = False
# Mask all the [PAD] tokens
mask = torch.logical_or(torch.tensor(mask)[None],
(inputs["attention_mask"] == 0))

start_logits[mask] = -10000
end_logits[mask] = -10000
```

```
start_probabilities = torch.nn.functional.softmax(start_logits, dim=-1)
end_probabilities = torch.nn.functional.softmax(end_logits, dim=-1)
```

```
candidates = []
for start_probs, end_probs in zip(start_probabilities, end_probabilities):
    scores = start_probs[:, None] * end_probs[None, :]
    idx = torch.triu(scores).argmax().item()

    start_idx = idx // scores.shape[1]
    end_idx = idx % scores.shape[1]
    score = scores[start_idx, end_idx].item()
    candidates.append((start_idx, end_idx, score))

print(candidates)
```

```
[(0, 18, 0.33867), (173, 184, 0.97149)]
```

```
for candidate, offset in zip(candidates, offsets):
    start_token, end_token, score = candidate
    start_char, _ = offset[start_token]
    _, end_char = offset[end_token]
    answer = long_context[start_char:end_char]
    result = {"answer": answer, "start": start_char, "end": end_char,
"score": score}
    print(result)
```

```
{'answer': '\n🤗 Transformers: State of the Art NLP', 'start': 0, 'end': 37, 'score': 0.33867}
{'answer': 'Jax, PyTorch and TensorFlow', 'start': 1892, 'end': 1919, 'score': 0.97149}
```

- **Normalization and pre-tokenization**

```
!pip install datasets evaluate transformers[sentencepiece]
```

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
print(type(tokenizer.backend_tokenizer))
```

```
<class 'tokenizers.Tokenizer'>
```

```
print(tokenizer.backend_tokenizer.normalizer.normalize_str("Héllò hôw are ü?"))
```

```
'hello how are u?'
```

```
tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_str("Hello, how are you?")
```

```
[('Hello', (0, 5)), (',', (5, 6)), ('how', (7, 10)), ('are', (11, 14)), ('you', (16, 19)), ('?', (19, 20))]
```

```
tokenizer = AutoTokenizer.from_pretrained("gpt2")
tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_str("Hello, how are you?")
```

```
[('Hello', (0, 5)), (',', (5, 6)), ('Ġhow', (6, 10)), ('Ġare', (10, 14)), ('Ġ', (14, 15)), ('Ġyou', (15, 19)),
 ('?', (19, 20))]
```

```
tokenizer = AutoTokenizer.from_pretrained("t5-small")
tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_str("Hello, how are you?")
```

```
[('_Hello,', (0, 6)), ('_how', (7, 10)), ('_are', (11, 14)), ('_you?', (16, 20))]
```

- **Byte-Pair Encoding tokenization**

```
!pip install datasets evaluate transformers[sentencepiece]
```

```
corpus = [
    "This is the Hugging Face Course.",
    "This chapter is about tokenization.",
    "This section shows several tokenizer algorithms.",
    "Hopefully, you will be able to understand how they are trained and generate tokens.",
]
```

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("gpt2")
```

```
from collections import defaultdict

word_freqs = defaultdict(int)

for text in corpus:
    words_with_offsets =
tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_str(text)
    new_words = [word for word, offset in words_with_offsets]
    for word in new_words:
        word_freqs[word] += 1

print(word_freqs)
```

```
defaultdict(int, {'This': 3, 'Ġis': 2, 'Ġthe': 1, 'ĠHugging': 1, 'ĠFace': 1, 'ĠCourse': 1, '.': 4,
'Ġchapter': 1,
    'Ġabout': 1, 'Ġtokenization': 1, 'Ġsection': 1, 'Ġshows': 1, 'Ġseveral': 1, 'Ġtokenizer': 1,
'Ġalgorithms': 1,
    'Hopefully': 1, ',': 1, 'Ġyou': 1, 'Ġwill': 1, 'Ġbe': 1, 'Ġable': 1, 'Ġto': 1, 'Ġunderstand':
1, 'Ġhow': 1,
    'Ġthey': 1, 'Ġare': 1, 'Ġtrained': 1, 'Ġand': 1, 'Ġgenerate': 1, 'Ġtokens': 1})
```

```
alphabet = []

for word in word_freqs.keys():
    for letter in word:
        if letter not in alphabet:
            alphabet.append(letter)
alphabet.sort()

print(alphabet)
```

```
[ ',', '.', 'C', 'F', 'H', 'T', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm', 'n',
'o', 'p', 'r', 's',
  't', 'u', 'v', 'w', 'y', 'z', 'Ġ']
```

```
vocab = ["<|endoftext|>"] + alphabet.copy()
```

```
splits = {word: [c for c in word] for word in word_freqs.keys()}
```

```
def compute_pair_freqs(splits):
    pair_freqs = defaultdict(int)
    for word, freq in word_freqs.items():
        split = splits[word]
        if len(split) == 1:
            continue
        for i in range(len(split) - 1):
            pair = (split[i], split[i + 1])
            pair_freqs[pair] += freq
    return pair_freqs
```

```
pair_freqs = compute_pair_freqs(splits)

for i, key in enumerate(pair_freqs.keys()):
    print(f"{key}: {pair_freqs[key]}")
    if i >= 5:
        break
```

```
('T', 'h'): 3
('h', 'i'): 3
('i', 's'): 5
('Ġ', 'i'): 2
('Ġ', 't'): 7
('t', 'h'): 3
```

```
best_pair = ""
max_freq = None

for pair, freq in pair_freqs.items():
    if max_freq is None or max_freq < freq:
        best_pair = pair
        max_freq = freq

print(best_pair, max_freq)
```

```
('Ġ', 't') 7
```

```
merges = {("Ġ", "t"): "Ġt"}
vocab.append("Ġt")
```

```
def merge_pair(a, b, splits):
    for word in word_freqs:
        split = splits[word]
        if len(split) == 1:
            continue

        i = 0
        while i < len(split) - 1:
            if split[i] == a and split[i + 1] == b:
                split = split[:i] + [a + b] + split[i + 2 :]
            else:
                i += 1
        splits[word] = split
    return splits
```

```
splits = merge_pair("Ġ", "t", splits)
print(splits["Ġtrained"])
```

```
['Ġt', 'r', 'a', 'i', 'n', 'e', 'd']
```

```
vocab_size = 50

while len(vocab) < vocab_size:
    pair_freqs = compute_pair_freqs(splits)
    best_pair = ""
    max_freq = None
    for pair, freq in pair_freqs.items():
        if max_freq is None or max_freq < freq:
            best_pair = pair
            max_freq = freq
    splits = merge_pair(*best_pair, splits)
    merges[best_pair] = best_pair[0] + best_pair[1]
    vocab.append(best_pair[0] + best_pair[1])
```

```
print(merges)
```

```
{('Ġ', 't'): 'Ġt', ('i', 's'): 'is', ('e', 'r'): 'er', ('Ġ', 'a'): 'Ġa', ('Ġt', 'o'): 'Ġto', ('e', 'n'): 'en',
 ('T', 'h'): 'Th', ('Th', 'is'): 'This', ('o', 'u'): 'ou', ('s', 'e'): 'se', ('Ġto', 'k'): 'Ġtok',
 ('Ġtok', 'en'): 'Ġtoken', ('n', 'd'): 'nd', ('Ġ', 'is'): 'Ġis', ('Ġt', 'h'): 'Ġth', ('Ġth', 'e'): 'Ġthe',
 ('i', 'n'): 'in', ('Ġa', 'b'): 'Ġab', ('Ġtoken', 'i'): 'Ġtokeni'}
```

```
print(vocab)
```

```
['<|endoftext|>', ',', '.', 'C', 'F', 'H', 'T', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k',
'l', 'm', 'n', 'o',
 'p', 'r', 's', 't', 'u', 'v', 'w', 'y', 'z', 'Ġ', 'Ġt', 'is', 'er', 'Ġa', 'Ġto', 'en', 'Th',
'This', 'ou', 'se',
 'Ġtok', 'Ġtoken', 'nd', 'Ġis', 'Ġth', 'Ġthe', 'in', 'Ġab', 'Ġtokeni']
```

```
def tokenize(text):
    pre_tokenize_result =
tokenizer._tokenizer.pre_tokenizer.pre_tokenize_str(text)
    pre_tokenized_text = [word for word, offset in pre_tokenize_result]
    splits = [[l for l in word] for word in pre_tokenized_text]
    for pair, merge in merges.items():
        for idx, split in enumerate(splits):
            i = 0
            while i < len(split) - 1:
                if split[i] == pair[0] and split[i + 1] == pair[1]:
                    split = split[:i] + [merge] + split[i + 2 :]
                else:
                    i += 1
            splits[idx] = split

    return sum(splits, [])
```

```
tokenize("This is not a token.")
```

```
['This', 'Ġis', 'Ġ', 'n', 'o', 't', 'Ġa', 'Ġtoken', '.']
```

- **WordPiece tokenization**

```
!pip install datasets evaluate transformers[sentencepiece]
```

```
corpus = [
    "This is the Hugging Face Course.",
    "This chapter is about tokenization.",
    "This section shows several tokenizer algorithms.",
    "Hopefully, you will be able to understand how they are trained and
generate tokens.",
]
```

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
```

```
from collections import defaultdict

word_freqs = defaultdict(int)
for text in corpus:
    words_with_offsets =
tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_str(text)
    new_words = [word for word, offset in words_with_offsets]
    for word in new_words:
        word_freqs[word] += 1

word_freqs
```

```
defaultdict(
    int, {'This': 3, 'is': 2, 'the': 1, 'Hugging': 1, 'Face': 1, 'Course': 1, '.': 4, 'chapter':
1, 'about': 1,
    'tokenization': 1, 'section': 1, 'shows': 1, 'several': 1, 'tokenizer': 1, 'algorithms': 1,
'Hopefully': 1,
    ',': 1, 'you': 1, 'will': 1, 'be': 1, 'able': 1, 'to': 1, 'understand': 1, 'how': 1, 'they':
1, 'are': 1,
    'trained': 1, 'and': 1, 'generate': 1, 'tokens': 1})
```

```
alphabet = []
for word in word_freqs.keys():
    if word[0] not in alphabet:
        alphabet.append(word[0])
    for letter in word[1:]:
        if f"##{letter}" not in alphabet:
            alphabet.append(f"##{letter}")

alphabet.sort()
alphabet

print(alphabet)
```

```
['##a', '##b', '##c', '##d', '##e', '##f', '##g', '##h', '##i', '##k', '##l', '##m', '##n', '##o',
'##p', '##r', '##s',
 '##t', '##u', '##v', '##w', '##y', '##z', ',', '.', 'C', 'F', 'H', 'T', 'a', 'b', 'c', 'g', 'h',
'i', 's', 't', 'u',
 'w', 'y']
```

```
vocab = ["[PAD]", "[UNK]", "[CLS]", "[SEP]", "[MASK]"] + alphabet.copy()
```

```
splits = {
    word: [c if i == 0 else f"##{c}" for i, c in enumerate(word)]
    for word in word_freqs.keys()
}
```

```python
def compute_pair_scores(splits):
    letter_freqs = defaultdict(int)
    pair_freqs = defaultdict(int)
    for word, freq in word_freqs.items():
        split = splits[word]
        if len(split) == 1:
            letter_freqs[split[0]] += freq
            continue
        for i in range(len(split) - 1):
            pair = (split[i], split[i + 1])
            letter_freqs[split[i]] += freq
            pair_freqs[pair] += freq
        letter_freqs[split[-1]] += freq

    scores = {
        pair: freq / (letter_freqs[pair[0]] * letter_freqs[pair[1]])
        for pair, freq in pair_freqs.items()
    }
    return scores
```

```python
pair_scores = compute_pair_scores(splits)
for i, key in enumerate(pair_scores.keys()):
    print(f"{key}: {pair_scores[key]}")
    if i >= 5:
        break
```

```
('T', '##h'): 0.125
('##h', '##i'): 0.03409090909090909
('##i', '##s'): 0.027272727272727
('i', '##s'): 0.1
('t', '##h'): 0.03571428571428571
('##h', '##e'): 0.011904761904761904
```

```
best_pair = ""
max_score = None
for pair, score in pair_scores.items():
    if max_score is None or max_score < score:
        best_pair = pair
        max_score = score

print(best_pair, max_score)
```

```
('a', '##b') 0.2
```

```
vocab.append("ab")
```

```
def merge_pair(a, b, splits):
    for word in word_freqs:
        split = splits[word]
        if len(split) == 1:
            continue
        i = 0
        while i < len(split) - 1:
            if split[i] == a and split[i + 1] == b:
                merge = a + b[2:] if b.startswith("##") else a + b
                split = split[:i] + [merge] + split[i + 2 :]
            else:
                i += 1
        splits[word] = split
    return splits
```

```
splits = merge_pair("a", "##b", splits)
splits["about"]
```

```
['ab', '##o', '##u', '##t']
```

```
vocab_size = 70
while len(vocab) < vocab_size:
    scores = compute_pair_scores(splits)
    best_pair, max_score = "", None
    for pair, score in scores.items():
        if max_score is None or max_score < score:
            best_pair = pair
            max_score = score
    splits = merge_pair(*best_pair, splits)
    new_token = (
        best_pair[0] + best_pair[1][2:]
        if best_pair[1].startswith("##")
        else best_pair[0] + best_pair[1]
    )
    vocab.append(new_token)
```

```
print(vocab)
```

```
['[PAD]', '[UNK]', '[CLS]', '[SEP]', '[MASK]', '##a', '##b', '##c', '##d', '##e', '##f', '##g',
'##h', '##i', '##k',
 '##l', '##m', '##n', '##o', '##p', '##r', '##s', '##t', '##u', '##v', '##w', '##y', '##z', ',',
'.', 'C', 'F', 'H',
 'T', 'a', 'b', 'c', 'g', 'h', 'i', 's', 't', 'u', 'w', 'y', 'ab', '##fu', 'Fa', 'Fac', '##ct',
'##ful', '##full', '##fully',
 'Th', 'ch', '##hm', 'cha', 'chap', 'chapt', '##thm', 'Hu', 'Hug', 'Hugg', 'sh', 'th', 'is',
'##thms', '##za', '##zat',
 '##ut']
```

```
def encode_word(word):
    tokens = []
    while len(word) > 0:
        i = len(word)
        while i > 0 and word[:i] not in vocab:
            i -= 1
        if i == 0:
            return ["[UNK]"]
        tokens.append(word[:i])
        word = word[i:]
        if len(word) > 0:
            word = f"##{word}"
    return tokens
```

```
print(encode_word("Hugging"))
print(encode_word("HOgging"))
```

```
['Hugg', '##i', '##n', '##g']
['[UNK]']
```

```
def tokenize(text):
    pre_tokenize_result =
tokenizer._tokenizer.pre_tokenizer.pre_tokenize_str(text)
    pre_tokenized_text = [word for word, offset in pre_tokenize_result]
    encoded_words = [encode_word(word) for word in pre_tokenized_text]
    return sum(encoded_words, [])
```

```
tokenize("This is the Hugging Face course!")
```

```
['Th', '##i', '##s', 'is', 'th', '##e', 'Hugg', '##i', '##n', '##g', 'Fac', '##e', 'c', '##o',
'##u', '##r', '##s',
 '##e', '[UNK]']
```

- **Unigram tokenization**

```
!pip install datasets evaluate transformers[sentencepiece]
```

```
corpus = [
    "This is the Hugging Face Course.",
    "This chapter is about tokenization.",
    "This section shows several tokenizer algorithms.",
    "Hopefully, you will be able to understand how they are trained and
generate tokens.",
]
```

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("xlnet-base-cased")
```

```
from collections import defaultdict

word_freqs = defaultdict(int)
for text in corpus:
    words_with_offsets =
tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_str(text)
    new_words = [word for word, offset in words_with_offsets]
    for word in new_words:
        word_freqs[word] += 1

word_freqs
```

```
char_freqs = defaultdict(int)
subwords_freqs = defaultdict(int)
for word, freq in word_freqs.items():
    for i in range(len(word)):
        char_freqs[word[i]] += freq
        # Loop through the subwords of length at least 2
        for j in range(i + 2, len(word) + 1):
            subwords_freqs[word[i:j]] += freq

# Sort subwords by frequency
sorted_subwords = sorted(subwords_freqs.items(), key=lambda x: x[1],
reverse=True)
sorted_subwords[:10]
```

```
[('_t', 7), ('is', 5), ('er', 5), ('_a', 5), ('_to', 4), ('to', 4), ('en', 4), ('_T', 3), ('_Th',
3), ('_Thi', 3)]
```

```
token_freqs = list(char_freqs.items()) + sorted_subwords[: 300 -
len(char_freqs)]
token_freqs = {token: freq for token, freq in token_freqs}
```

```
from math import log

total_sum = sum([freq for token, freq in token_freqs.items()])
model = {token: -log(freq / total_sum) for token, freq in
token_freqs.items()}
```

```python
def encode_word(word, model):
    best_segmentations = [{"start": 0, "score": 1}] + [
        {"start": None, "score": None} for _ in range(len(word))
    ]
    for start_idx in range(len(word)):
        # This should be properly filled by the previous steps of the loop
        best_score_at_start = best_segmentations[start_idx]["score"]
        for end_idx in range(start_idx + 1, len(word) + 1):
            token = word[start_idx:end_idx]
            if token in model and best_score_at_start is not None:
                score = model[token] + best_score_at_start
                # If we have found a better segmentation ending at
end_idx, we update
                if (
                    best_segmentations[end_idx]["score"] is None
                    or best_segmentations[end_idx]["score"] > score
                ):
                    best_segmentations[end_idx] = {"start": start_idx,
"score": score}

    segmentation = best_segmentations[-1]
    if segmentation["score"] is None:
        # We did not find a tokenization of the word -> unknown
        return ["<unk>"], None

    score = segmentation["score"]
    start = segmentation["start"]
    end = len(word)
    tokens = []
    while start != 0:
        tokens.insert(0, word[start:end])
        next_start = best_segmentations[start]["start"]
        end = start
        start = next_start
    tokens.insert(0, word[start:end])
    return tokens, score
```

```python
print(encode_word("Hopefully", model))
print(encode_word("This", model))
```

```
(['H', 'o', 'p', 'e', 'f', 'u', 'll', 'y'], 41.5157494601402)
(['This'], 6.288267030694535)
```

```python
def compute_loss(model):
    loss = 0
    for word, freq in word_freqs.items():
        _, word_loss = encode_word(word, model)
        loss += freq * word_loss
    return loss
```

```python
compute_loss(model)
```

```
413.10377642940875
```

```python
import copy


def compute_scores(model):
    scores = {}
    model_loss = compute_loss(model)
    for token, score in model.items():
        # We always keep tokens of length 1
        if len(token) == 1:
            continue
        model_without_token = copy.deepcopy(model)
        _ = model_without_token.pop(token)
        scores[token] = compute_loss(model_without_token) - model_loss
    return scores
```

```python
scores = compute_scores(model)
print(scores["ll"])
print(scores["his"])
```

```
6.376412403623874
0.0
```

```
percent_to_remove = 0.1
while len(model) > 100:
    scores = compute_scores(model)
    sorted_scores = sorted(scores.items(), key=lambda x: x[1])
    # Remove percent_to_remove tokens with the lowest scores.
    for i in range(int(len(model) * percent_to_remove)):
        _ = token_freqs.pop(sorted_scores[i][0])

    total_sum = sum([freq for token, freq in token_freqs.items()])
    model = {token: -log(freq / total_sum) for token, freq in
token_freqs.items()}
```

```
def tokenize(text, model):
    words_with_offsets =
tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_str(text)
    pre_tokenized_text = [word for word, offset in words_with_offsets]
    encoded_words = [encode_word(word, model)[0] for word in
pre_tokenized_text]
    return sum(encoded_words, [])


tokenize("This is the Hugging Face course.", model)
```

```
['_This', '_is', '_the', '_Hugging', '_Face', '_', 'c', 'ou', 'r', 's', 'e', '.']
```

- **Building a tokenizer, block by block**

```
!pip install datasets evaluate transformers[sentencepiece]
```

```
from datasets import load_dataset

dataset = load_dataset("wikitext", name="wikitext-2-raw-v1",
split="train")


def get_training_corpus():
    for i in range(0, len(dataset), 1000):
        yield dataset[i : i + 1000]["text"]
```

```
with open("wikitext-2.txt", "w", encoding="utf-8") as f:
    for i in range(len(dataset)):
        f.write(dataset[i]["text"] + "\n")
```

```
from tokenizers import (
    decoders,
    models,
    normalizers,
    pre_tokenizers,
    processors,
    trainers,
    Tokenizer,
)

tokenizer = Tokenizer(models.WordPiece(unk_token="[UNK]"))
```

```
tokenizer.normalizer = normalizers.BertNormalizer(lowercase=True)
```

```
tokenizer.normalizer = normalizers.Sequence(
    [normalizers.NFD(), normalizers.Lowercase(),
normalizers.StripAccents()]
)
```

```
print(tokenizer.normalizer.normalize_str("Héllò hôw are ü?"))
```

```
hello how are u?
```

```
tokenizer.pre_tokenizer = pre_tokenizers.BertPreTokenizer()
```

```
tokenizer.pre_tokenizer = pre_tokenizers.Whitespace()
```

```
tokenizer.pre_tokenizer.pre_tokenize_str("Let's test my pre-tokenizer.")
```

```
[('Let', (0, 3)), ("'", (3, 4)), ('s', (4, 5)), ('test', (6, 10)), ('my', (11, 13)), ('pre', (14,
17)),
 ('-', (17, 18)), ('tokenizer', (18, 27)), ('.', (27, 28))]
```

```
pre_tokenizer = pre_tokenizers.WhitespaceSplit()
pre_tokenizer.pre_tokenize_str("Let's test my pre-tokenizer.")
```

```
[("Let's", (0, 5)), ('test', (6, 10)), ('my', (11, 13)), ('pre-tokenizer.', (14, 28))]
```

```
pre_tokenizer = pre_tokenizers.Sequence(
    [pre_tokenizers.WhitespaceSplit(), pre_tokenizers.Punctuation()]
)
pre_tokenizer.pre_tokenize_str("Let's test my pre-tokenizer.")
```

```
[('Let', (0, 3)), ("'", (3, 4)), ('s', (4, 5)), ('test', (6, 10)), ('my', (11, 13)), ('pre', (14, 17)),
 ('-', (17, 18)), ('tokenizer', (18, 27)), ('.', (27, 28))]
```

```
special_tokens = ["[UNK]", "[PAD]", "[CLS]", "[SEP]", "[MASK]"]
trainer = trainers.WordPieceTrainer(vocab_size=25000,
special_tokens=special_tokens)
```

```
tokenizer.train_from_iterator(get_training_corpus(), trainer=trainer)
```

```
tokenizer.model = models.WordPiece(unk_token="[UNK]")
tokenizer.train(["wikitext-2.txt"], trainer=trainer)
```

```
encoding = tokenizer.encode("Let's test this tokenizer.")
print(encoding.tokens)
```

```
['let', "'", 's', 'test', 'this', 'tok', '##eni', '##zer', '.']
```

```
cls_token_id = tokenizer.token_to_id("[CLS]")
sep_token_id = tokenizer.token_to_id("[SEP]")
print(cls_token_id, sep_token_id)
```

```
(2, 3)
```

```
tokenizer.post_processor = processors.TemplateProcessing(
    single=f"[CLS]:0 $A:0 [SEP]:0",
    pair=f"[CLS]:0 $A:0 [SEP]:0 $B:1 [SEP]:1",
    special_tokens=[("[CLS]", cls_token_id), ("[SEP]", sep_token_id)],
)
```

```
encoding = tokenizer.encode("Let's test this tokenizer.")
print(encoding.tokens)
```

```
['[CLS]', 'let', "'", 's', 'test', 'this', 'tok', '##eni', '##zer', '.', '[SEP]']
```

```
encoding = tokenizer.encode("Let's test this tokenizer...", "on a pair of
sentences.")
print(encoding.tokens)
print(encoding.type_ids)
```

```
['[CLS]', 'let', "'", 's', 'test', 'this', 'tok', '##eni', '##zer', '...', '[SEP]', 'on', 'a',
'pair', 'of', 'sentences', '.', '[SEP]']
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
```

```
tokenizer.decoder = decoders.WordPiece(prefix="##")
```

```
tokenizer.decode(encoding.ids)
```

```
"let's test this tokenizer... on a pair of sentences."
```

```
tokenizer.save("tokenizer.json")
```

```
new_tokenizer = Tokenizer.from_file("tokenizer.json")
```

```
from transformers import PreTrainedTokenizerFast

wrapped_tokenizer = PreTrainedTokenizerFast(
    tokenizer_object=tokenizer,
    # tokenizer_file="tokenizer.json", # You can load from the tokenizer
file, alternatively
    unk_token="[UNK]",
    pad_token="[PAD]",
    cls_token="[CLS]",
    sep_token="[SEP]",
    mask_token="[MASK]",
)
```

```
from transformers import BertTokenizerFast

wrapped_tokenizer = BertTokenizerFast(tokenizer_object=tokenizer)
```

```
tokenizer = Tokenizer(models.BPE())
```

```
tokenizer.pre_tokenizer = pre_tokenizers.ByteLevel(add_prefix_space=False)
```

```
tokenizer.pre_tokenizer.pre_tokenize_str("Let's test pre-tokenization!")
```

```
[('Let', (0, 3)), ("'s", (3, 5)), ('Ġtest', (5, 10)), ('Ġpre', (10, 14)), ('-', (14, 15)),
 ('tokenization', (15, 27)), ('!', (27, 28))]
```

```
trainer = trainers.BpeTrainer(vocab_size=25000,
special_tokens=["<|endoftext|>"])
tokenizer.train_from_iterator(get_training_corpus(), trainer=trainer)
```

```
tokenizer.model = models.BPE()
tokenizer.train(["wikitext-2.txt"], trainer=trainer)
```

```
encoding = tokenizer.encode("Let's test this tokenizer.")
print(encoding.tokens)
```

```
['L', 'et', "'", 's', 'Ġtest', 'Ġthis', 'Ġto', 'ken', 'izer', '.']
```

```
tokenizer.post_processor = processors.ByteLevel(trim_offsets=False)
```

```
sentence = "Let's test this tokenizer."
encoding = tokenizer.encode(sentence)
start, end = encoding.offsets[4]
sentence[start:end]
```

```
' test'
```

```
tokenizer.decoder = decoders.ByteLevel()
```

```
tokenizer.decode(encoding.ids)
```

```
"Let's test this tokenizer."
```

```
from transformers import PreTrainedTokenizerFast

wrapped_tokenizer = PreTrainedTokenizerFast(
    tokenizer_object=tokenizer,
    bos_token="<|endoftext|>",
    eos_token="<|endoftext|>",
)


from transformers import GPT2TokenizerFast

wrapped_tokenizer = GPT2TokenizerFast(tokenizer_object=tokenizer)


tokenizer = Tokenizer(models.Unigram())


from tokenizers import Regex

tokenizer.normalizer = normalizers.Sequence(
    [
        normalizers.Replace("``", '"'),
        normalizers.Replace("''", '"'),
        normalizers.NFKD(),
        normalizers.StripAccents(),
        normalizers.Replace(Regex(" {2,}"), " "),
    ]
)


tokenizer.pre_tokenizer = pre_tokenizers.Metaspace()


tokenizer.pre_tokenizer.pre_tokenize_str("Let's test the pre-tokenizer!")
```

```
[("_Let's", (0, 5)), ('_test', (5, 10)), ('_the', (10, 14)), ('_pre-tokenizer!', (14, 29))]
```

```
special_tokens = ["<cls>", "<sep>", "<unk>", "<pad>", "<mask>", "<s>",
"</s>"]
trainer = trainers.UnigramTrainer(
    vocab_size=25000, special_tokens=special_tokens, unk_token="<unk>"
)
tokenizer.train_from_iterator(get_training_corpus(), trainer=trainer)
```

```
tokenizer.model = models.Unigram()
tokenizer.train(["wikitext-2.txt"], trainer=trainer)
```

```
encoding = tokenizer.encode("Let's test this tokenizer.")
print(encoding.tokens)
```

```
['_Let', "'", 's', '_test', '_this', '_to', 'ken', 'izer', '.']
```

```
cls_token_id = tokenizer.token_to_id("<cls>")
sep_token_id = tokenizer.token_to_id("<sep>")
print(cls_token_id, sep_token_id)
```

```
0 1
```

```
tokenizer.post_processor = processors.TemplateProcessing(
    single="$A:0 <sep>:0 <cls>:2",
    pair="$A:0 <sep>:0 $B:1 <sep>:1 <cls>:2",
    special_tokens=[("<sep>", sep_token_id), ("<cls>", cls_token_id)],
)
```

```
encoding = tokenizer.encode("Let's test this tokenizer...", "on a pair of sentences!")
print(encoding.tokens)
print(encoding.type_ids)
```

```
['_Let', "'", 's', '_test', '_this', '_to', 'ken', 'izer', '.', '.', '.', '<sep>', '_', 'on', '_', 'a', '_pair',
  '_of', '_sentence', 's', '!', '<sep>', '<cls>']
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2]
```

```
tokenizer.decoder = decoders.Metaspace()
```

```
from transformers import PreTrainedTokenizerFast

wrapped_tokenizer = PreTrainedTokenizerFast(
    tokenizer_object=tokenizer,
    bos_token="<s>",
    eos_token="</s>",
    unk_token="<unk>",
    pad_token="<pad>",
    cls_token="<cls>",
    sep_token="<sep>",
    mask_token="<mask>",
    padding_side="left",
)


from transformers import XLNetTokenizerFast

wrapped_tokenizer = XLNetTokenizerFast(tokenizer_object=tokenizer.
```

7. Main NLP Tasks

- **Token Classification**

```
!pip install datasets evaluate transformers[sentencepiece]
!pip install accelerate
# To run the training on TPU, you will need to uncomment the following
line:
# !pip install cloud-tpu-client==0.10 torch==1.9.0
https://storage.googleapis.com/tpu-pytorch/wheels/torch_xla-1.9-cp37-
cp37m-linux_x86_64.whl
!apt install git-lfs
```

```
    Successfully installed datasets-3.2.0 dill-0.3.8 evaluate-0.4.3 fsspec-2024.9.0 mul
    Requirement already satisfied: accelerate in /usr/local/lib/python3.10/dist-package
    Requirement already satisfied: numpy<3.0.0,>=1.17 in /usr/local/lib/python3.10/dist
    Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-pa
    Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (f
    Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (f
    Requirement already satisfied: torch>=1.10.0 in /usr/local/lib/python3.10/dist-pack
    Requirement already satisfied: huggingface-hub>=0.21.0 in /usr/local/lib/python3.10
    Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.10/dist
    Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages
    Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-p
    Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages
    Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-packa
    Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3
    Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages
    Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (f
    Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-pack
    Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist
    Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-pa
    Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.1
    Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packa
    Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist
    Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist
    Reading package lists... Done
    Building dependency tree... Done
    Reading state information... Done
    git-lfs is already the newest version (3.0.2-1ubuntu0.3).
    0 upgraded, 0 newly installed, 0 to remove and 49 not upgraded.
```

```
!git config --global user.email "you@example.com"
!git config --global user.name "Your Name"


from huggingface_hub import notebook_login

notebook_login()


from datasets import load_dataset

raw_datasets = load_dataset("conll2003")
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: 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://hugging
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 da
  warnings.warn(
README.md: 100% 12.3k/12.3k [00:00<00:00, 419kB/s]
conll2003.py: 100% 9.57k/9.57k [00:00<00:00, 207kB/s]
The repository for conll2003 contains custom code which must be executed to correctly load the
You can avoid this prompt in future by passing the argument `trust_remote_code=True`.

Do you wish to run the custom code? [y/N] y
Downloading data: 100% 983k/983k [00:00<00:00, 4.57MB/s]
Generating train split: 100% 14041/14041 [00:04<00:00, 3208.32 examples/s]
Generating validation split: 100% 3250/3250 [00:00<00:00, 3464.62 examples/s]
Generating test split: 100% 3453/3453 [00:00<00:00, 3910.41 examples/s]
```

```
raw_datasets
```

```
DatasetDict({
    train: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 14041
    })
    validation: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 3250
    })
    test: Dataset({
        features: ['id', 'tokens', 'pos_tags', 'chunk_tags', 'ner_tags'],
        num_rows: 3453
    })
})
```

```
raw_datasets["train"][0]["tokens"]
```

```
['EU', 'rejects', 'German', 'call', 'to', 'boycott', 'British', 'lamb', '.']
```

```
raw_datasets["train"][0]["ner_tags"]
```

```
[3, 0, 7, 0, 0, 0, 7, 0, 0]
```

```
ner_feature = raw_datasets["train"].features["ner_tags"]
ner_feature
```

```
Sequence(feature=ClassLabel(names=['O', 'B-PER', 'I-PER', 'B-ORG', 'I-ORG', 'B-LOC', 'I-LOC', 'B-MISC', 'I-MISC'], id=None), length=-1, id=None)
```

```
label_names = ner_feature.feature.names
label_names
```

```
['O', 'B-PER', 'I-PER', 'B-ORG', 'I-ORG', 'B-LOC', 'I-LOC', 'B-MISC', 'I-MISC']
```

```
words = raw_datasets["train"][0]["tokens"]
labels = raw_datasets["train"][0]["ner_tags"]
line1 = ""
line2 = ""
for word, label in zip(words, labels):
    full_label = label_names[label]
    max_length = max(len(word), len(full_label))
    line1 += word + " " * (max_length - len(word) + 1)
    line2 += full_label + " " * (max_length - len(full_label) + 1)

print(line1)
print(line2)
```

```
from transformers import AutoTokenizer

model_checkpoint = "bert-base-cased"
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
```

```
The cache for model files in Transformers v4.22.0 has been updated. Migrating your old cache. This
0/0 [00:00<?, ?it/s]
tokenizer_config.json: 100% 49.0/49.0 [00:00<00:00, 2.38kB/s]
config.json: 100% 570/570 [00:00<00:00, 30.5kB/s]
vocab.txt: 100% 213k/213k [00:00<00:00, 3.10MB/s]
tokenizer.json: 100% 436k/436k [00:00<00:00, 5.38MB/s]
```

```
tokenizer.is_fast
```

```
True
```

```
inputs = tokenizer(raw_datasets["train"][0]["tokens"],
is_split_into_words=True)
inputs.tokens()
```

```
['[CLS]',
 'EU',
 'rejects',
 'German',
 'call',
 'to',
 'boycott',
 'British',
 'la',
 '##mb',
 '.',
 '[SEP]']
```

```
inputs.word_ids()
```

```
[None, 0, 1, 2, 3, 4, 5, 6, 7, 7, 8, None]
```

```python
def align_labels_with_tokens(labels, word_ids):
    new_labels = []
    current_word = None
    for word_id in word_ids:
        if word_id != current_word:
            # Start of a new word!
            current_word = word_id
            label = -100 if word_id is None else labels[word_id]
            new_labels.append(label)
        elif word_id is None:
            # Special token
            new_labels.append(-100)
        else:
            # Same word as previous token
            label = labels[word_id]
            # If the label is B-XXX we change it to I-XXX
            if label % 2 == 1:
                label += 1
            new_labels.append(label)

    return new_labels


labels = raw_datasets["train"][0]["ner_tags"]
word_ids = inputs.word_ids()
print(labels)
print(align_labels_with_tokens(labels, word_ids))


def tokenize_and_align_labels(examples):
    tokenized_inputs = tokenizer(
        examples["tokens"], truncation=True, is_split_into_words=True
    )
    all_labels = examples["ner_tags"]
    new_labels = []
    for i, labels in enumerate(all_labels):
        word_ids = tokenized_inputs.word_ids(i)
        new_labels.append(align_labels_with_tokens(labels, word_ids))

    tokenized_inputs["labels"] = new_labels
    return tokenized_inputs
```

```
tokenized_datasets = raw_datasets.map(
    tokenize_and_align_labels,
    batched=True,
    remove_columns=raw_datasets["train"].column_names,
)
```

```
Map: 100% 14041/14041 [00:03<00:00, 5619.02 examples/s]
Map: 100% 3250/3250 [00:00<00:00, 5491.86 examples/s]
Map: 100% 3453/3453 [00:00<00:00, 6689.11 examples/s]
```

```
from transformers import DataCollatorForTokenClassification

data_collator = DataCollatorForTokenClassification(tokenizer=tokenizer)
```

```
batch = data_collator([tokenized_datasets["train"][i] for i in range(2)])
batch["labels"]
```

```
tensor([[-100,    3,    0,    7,    0,    0,    0,    7,    0,    0,    0, -100],
        [-100,    1,    2, -100, -100, -100, -100, -100, -100, -100, -100, -100]])
```

```
for i in range(2):
    print(tokenized_datasets["train"][i]["labels"])
```

```
!pip install seqeval
```

```
import evaluate

metric = evaluate.load("seqeval")
```

```
Downloading builder script: 100% 6.34k/6.34k [00:00<00:00, 77.7kB/s]
```

```
labels = raw_datasets["train"][0]["ner_tags"]
labels = [label_names[i] for i in labels]
labels
```

```
['B-ORG', 'O', 'B-MISC', 'O', 'O', 'O', 'B-MISC', 'O', 'O']
```

```
predictions = labels.copy()
predictions[2] = "O"
metric.compute(predictions=[predictions], references=[labels])
```

```
{'MISC': {'precision': 1.0,
  'recall': 0.5,
  'f1': 0.6666666666666666,
  'number': 2},
 'ORG': {'precision': 1.0, 'recall': 1.0, 'f1': 1.0, 'number': 1},
 'overall_precision': 1.0,
 'overall_recall': 0.6666666666666666,
 'overall_f1': 0.8,
 'overall_accuracy': 0.8888888888888888}
```

```
import numpy as np


def compute_metrics(eval_preds):
    logits, labels = eval_preds
    predictions = np.argmax(logits, axis=-1)

    # Remove ignored index (special tokens) and convert to labels
    true_labels = [[label_names[l] for l in label if l != -100] for label
in labels]
    true_predictions = [
        [label_names[p] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]
    all_metrics = metric.compute(predictions=true_predictions,
references=true_labels)
    return {
        "precision": all_metrics["overall_precision"],
        "recall": all_metrics["overall_recall"],
        "f1": all_metrics["overall_f1"],
        "accuracy": all_metrics["overall_accuracy"],
    }


id2label = {i: label for i, label in enumerate(label_names)}
label2id = {v: k for k, v in id2label.items()}
```

```python
from transformers import AutoModelForTokenClassification

model = AutoModelForTokenClassification.from_pretrained(
    model_checkpoint,
    id2label=id2label,
    label2id=label2id,
)
```

```
model.safetensors: 100% 436M/436M [00:07<00:00, 58.4MB/s]
Some weights of BertForTokenClassification were not initialized from the model checkpoint at bert-b
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
```

```python
model.config.num_labels
```

```
9
```

```python
from huggingface_hub import notebook_login

notebook_login()
```

```python
from transformers import TrainingArguments

args = TrainingArguments(
    "bert-finetuned-ner",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    num_train_epochs=3,
    weight_decay=0.01,
    push_to_hub=True,
)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1575: FutureWarning: `evaluat
  warnings.warn(
```

```python
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=data_collator,
    compute_metrics=compute_metrics,
    tokenizer=tokenizer,
)
trainer.train()
```

```
<ipython-input-41-ddd9b7f8600f>:3: FutureWarning: `tokenizer` is deprecated and will be removed in
  trainer = Trainer(
```

[20/5268 01:29 < 7:13:07, 0.20 it/s, Epoch 0.01/3]

Epoch	Training Loss	Validation Loss

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-41-ddd9b7f8600f> in <cell line: 12>()
     10     tokenizer=tokenizer,
     11 )
---> 12 trainer.train()

                                 24 frames
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/linear.py in forward(self, input)
    123
    124     def forward(self, input: Tensor) -> Tensor:
--> 125         return F.linear(input, self.weight, self.bias)
    126
    127     def extra_repr(self) -> str:

KeyboardInterrupt:
```

code ini error dikarenakan trainernya tidak saya selesaikan karena membutuhkan waktu yang cukup lama -+7 jam

```
trainer.push_to_hub(commit_message="Training complete")
```

```
model.safetensors: 100% 431M/431M [00:08<00:00, 54.6MB/s]
events.out.tfevents.1735994322.6734a527aa14.397.0: 100% 5.39k/5.39k [00:00<00:00, 29.0kB/s]
events.out.tfevents.1735994627.6734a527aa14.397.1: 100% 5.39k/5.39k [00:00<00:00, 31.4kB/s]
Upload 6 LFS files: 100% 6/6 [00:08<00:00, 8.50s/it]
events.out.tfevents.1735995048.6734a527aa14.397.2: 100% 5.39k/5.39k [00:00<00:00, 31.4kB/s]
events.out.tfevents.1735995083.6734a527aa14.397.3: 100% 5.39k/5.39k [00:00<00:00, 31.9kB/s]
training_args.bin: 100% 5.30k/5.30k [00:00<00:00, 65.2kB/s]
CommitInfo(commit_url='https://huggingface.co/ikyyy1/bert-finetuned-ner/commit/b55369d989835c9ae007ccd69191bcbfa7af0cf4', commit_message='Training complete', commit_description='', oid='b55369d989835c9ae007ccd69191bcbfa7af0cf4', pr_url=None, repo_url=RepoUrl('https://huggingface.co/ikyyy1/bert-finetuned-ner', endpoint='https://huggingface.co', repo_type='model', repo_id='ikyyy1/bert-finetuned-ner'), pr_revision=None, pr_num=None)
```

```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(
    tokenized_datasets["train"],
    shuffle=True,
    collate_fn=data_collator,
    batch_size=8,
)
eval_dataloader = DataLoader(
    tokenized_datasets["validation"], collate_fn=data_collator,
batch_size=8
)


model = AutoModelForTokenClassification.from_pretrained(
    model_checkpoint,
    id2label=id2label,
    label2id=label2id,
)
```

```
Some weights of BertForTokenClassification were not initialized from the model checkpoint at bert-l
You should probably TRAIN this model on a down-stream task to be able to use it for predictions an
```

```
from torch.optim import AdamW

optimizer = AdamW(model.parameters(), lr=2e-5)
```

```python
from accelerate import Accelerator

accelerator = Accelerator()
model, optimizer, train_dataloader, eval_dataloader = accelerator.prepare(
    model, optimizer, train_dataloader, eval_dataloader
)


from transformers import get_scheduler

num_train_epochs = 3
num_update_steps_per_epoch = len(train_dataloader)
num_training_steps = num_train_epochs * num_update_steps_per_epoch

lr_scheduler = get_scheduler(
    "linear",
    optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=num_training_steps,
)


from huggingface_hub import Repository, get_full_repo_name

model_name = "bert-finetuned-ner-accelerate"
repo_name = get_full_repo_name(model_name)
repo_name
```

```
'ikyyy1/bert-finetuned-ner-accelerate'
```

```
output_dir = "bert-finetuned-ner-accelerate"
repo = Repository("<bert-finetuned-ner>", clone_from="git clone
https://huggingface.co/ikyyy1/bert-finetuned-ner")
```

```
For more details, please read https://huggingface.co/docs/huggingface_hub/concepts/git_vs_http.
  warnings.warn(warning_message, FutureWarning)
Cloning https://huggingface.co/ikyyy1/bert-finetuned-ner into local empty directory.
WARNING:huggingface_hub.repository:Cloning https://huggingface.co/ikyyy1/bert-finetuned-ner into l
Download file model.safetensors: 100% 411M/411M [02:03<00:00, 284kB/s]
Download file runs/Jan04_12-32-
31_6734a527aa14/events.out.tfevents.1735995048.6734a527aa14.397.2: 100% 5.27k/5.27k [02:03<?, ?B/s]
Download file runs/Jan04_12-32-
31_6734a527aa14/events.out.tfevents.1735995083.6734a527aa14.397.3: 100% 5.27k/5.27k [02:03<?, ?B/s]
Download file training_args.bin: 100% 5.18k/5.18k [02:03<?, ?B/s]
Download file runs/Jan04_12-32-
31_6734a527aa14/events.out.tfevents.1735994322.6734a527aa14.397.0: 100% 5.27k/5.27k [02:03<?, ?B/s]
Download file runs/Jan04_12-32-
31_6734a527aa14/events.out.tfevents.1735994627.6734a527aa14.397.1: 100% 5.27k/5.27k [02:03<?, ?B/s]
Clean file runs/Jan04_12-32-
31_6734a527aa14/events.out.tfevents.1735995048.6734a527aa14.397.2: 100% 5.27k/5.27k [02:03<00:00, 35.5B/s]
Clean file training_args.bin: 100% 5.18k/5.18k [02:02<00:00, 34.8B/s]
Clean file runs/Jan04_12-32-
31_6734a527aa14/events.out.tfevents.1735995083.6734a527aa14.397.3: 100% 5.27k/5.27k [02:02<00:00, 38.9kB/s]
Clean file runs/Jan04_12-32-
31_6734a527aa14/events.out.tfevents.1735994322.6734a527aa14.397.0: 100% 5.27k/5.27k [02:02<00:00, 35.6B/s]
Clean file runs/Jan04_12-32-
31_6734a527aa14/events.out.tfevents.1735994627.6734a527aa14.397.1: 100% 5.27k/5.27k [02:02<00:00, 35.6B/s]
Clean file model.safetensors: 100% 411M/411M [00:57<00:00, 8.68MB/s]
```

```
def postprocess(predictions, labels):
    predictions = predictions.detach().cpu().clone().numpy()
    labels = labels.detach().cpu().clone().numpy()

    # Remove ignored index (special tokens) and convert to labels
    true_labels = [[label_names[l] for l in label if l != -100] for label
in labels]
    true_predictions = [
        [label_names[p] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]
    return true_labels, true_predictions
```

```
from tqdm.auto import tqdm
import torch

progress_bar = tqdm(range(num_training_steps))

for epoch in range(num_train_epochs):
    # Training
    model.train()
    for batch in train_dataloader:
        outputs = model(**batch)
        loss = outputs.loss
        accelerator.backward(loss)

        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()
        progress_bar.update(1)

    # Evaluation
    model.eval()
    for batch in eval_dataloader:
        with torch.no_grad():
            outputs = model(**batch)

        predictions = outputs.logits.argmax(dim=-1)
        labels = batch["labels"]

        # Necessary to pad predictions and labels for being gathered
        predictions = accelerator.pad_across_processes(predictions, dim=1,
pad_index=-100)
        labels = accelerator.pad_across_processes(labels, dim=1,
pad_index=-100)

        predictions_gathered = accelerator.gather(predictions)
        labels_gathered = accelerator.gather(labels)

        true_predictions, true_labels = postprocess(predictions_gathered,
labels_gathered)
        metric.add_batch(predictions=true_predictions,
references=true_labels)

    results = metric.compute()
    print(
        f"epoch {epoch}:",
        {
```

```
            key: results[f"overall_{key}"]
            for key in ["precision", "recall", "f1", "accuracy"]
        },
    )

    # Save and upload
    accelerator.wait_for_everyone()
    unwrapped_model = accelerator.unwrap_model(model)
    unwrapped_model.save_pretrained(output_dir,
save_function=accelerator.save)
    if accelerator.is_main_process:
        tokenizer.save_pretrained(output_dir)
        repo.push_to_hub(
            commit_message=f"Training in progress epoch {epoch}",
blocking=False
        )
```

```
2%  114/5268 [09:22<6:32:40,  4.57s/it]
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-54-db1270503d5a> in <cell line: 6>()
      8     model.train()
      9     for batch in train_dataloader:
---> 10         outputs = model(**batch)
     11         loss = outputs.loss
     12         accelerator.backward(loss)

                                 19 frames
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/linear.py in forward(self, input)
    123
    124     def forward(self, input: Tensor) -> Tensor:
--> 125         return F.linear(input, self.weight, self.bias)
    126
    127     def extra_repr(self) -> str:

KeyboardInterrupt:
```

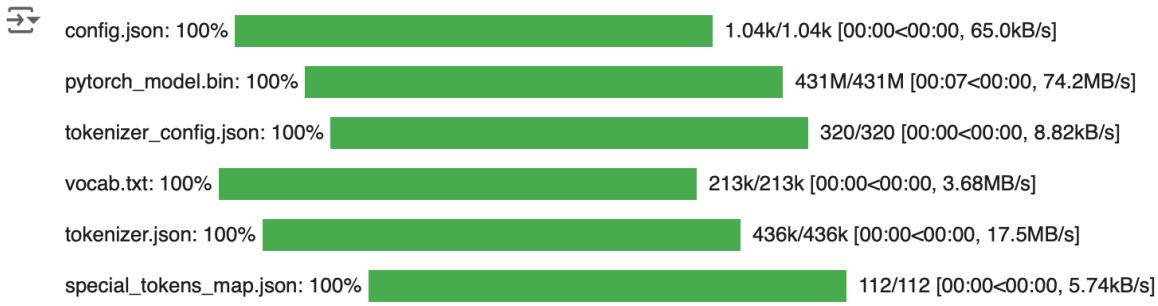
code error dikarenakan tidak selesai training karena membutuhkan waktu yang cukup lama sekitar 6 jam

```
[ ] accelerator.wait_for_everyone()
```

```
accelerator.wait_for_everyone()
unwrapped_model = accelerator.unwrap_model(model)
unwrapped_model.save_pretrained(output_dir,
save_function=accelerator.save)
```

```
from transformers import pipeline

# Replace this with your own checkpoint
model_checkpoint = "huggingface-course/bert-finetuned-ner"
token_classifier = pipeline(
    "token-classification", model=model_checkpoint,
aggregation_strategy="simple"
)
token_classifier("My name is Sylvain and I work at Hugging Face in
Brooklyn.")
```




```
Device set to use cpu
[{'entity_group': 'PER',
  'score': 0.9988506,
  'word': 'Sylvain',
  'start': 11,
  'end': 18},
 {'entity_group': 'ORG',
  'score': 0.9647625,
  'word': 'Hugging Face',
  'start': 33,
  'end': 45},
 {'entity_group': 'LOC',
  'score': 0.9986118,
  'word': 'Brooklyn',
  'start': 49,
  'end': 57}]
```

- **Fine-tuning a masked language model**

```
!pip install datasets evaluate transformers[sentencepiece]
!pip install accelerate
# To run the training on TPU, you will need to uncomment the following
line:
# !pip install cloud-tpu-client==0.10 torch==1.9.0
https://storage.googleapis.com/tpu-pytorch/wheels/torch_xla-1.9-cp37-
cp37m-linux_x86_64.whl
!apt install git-lfs
```

```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist
Requirement already satisfied: torch>=1.10.0 in /usr/local/lib/python3.
Requirement already satisfied: huggingface-hub>=0.21.0 in /usr/local/li
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/pyt
Requirement already satisfied: filelock in /usr/local/lib/python3.10/di
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/pytho
Requirement already satisfied: requests in /usr/local/lib/python3.10/di
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.1
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local
Requirement already satisfied: networkx in /usr/local/lib/python3.10/di
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist
Collecting sympy==1.13.1 (from torch>=1.10.0->accelerate)
  Downloading sympy-1.13.1-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/pyt
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/l
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.1
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/pyt
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/pyt
Downloading sympy-1.13.1-py3-none-any.whl (6.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.2/6.2 MB 21.7 MB/s eta 0:
Installing collected packages: sympy
  Attempting uninstall: sympy
    Found existing installation: sympy 1.13.3
    Uninstalling sympy-1.13.3:
      Successfully uninstalled sympy-1.13.3
Successfully installed sympy-1.13.1
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
git-lfs is already the newest version (3.0.2-1ubuntu0.3).
0 upgraded, 0 newly installed, 0 to remove and 49 not upgraded.
```

```
!git config --global user.email "m.rizkyf400@gmail.com"
!git config --global user.name "rizkyy3"


from huggingface_hub import notebook_login

notebook_login()
```

```
pip install --upgrade sympy
```

```
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (1.13.1)
Collecting sympy
  Using cached sympy-1.13.3-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (f
Using cached sympy-1.13.3-py3-none-any.whl (6.2 MB)
Installing collected packages: sympy
  Attempting uninstall: sympy
    Found existing installation: sympy 1.13.1
    Uninstalling sympy-1.13.1:
      Successfully uninstalled sympy-1.13.1
ERROR: pip's dependency resolver does not currently take into account all the packages that are
torch 2.5.1+cu121 requires sympy==1.13.1; python_version >= "3.9", but you have sympy 1.13.3 wh
Successfully installed sympy-1.13.3
```

```
from transformers import AutoModelForMaskedLM

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

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: 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://huggi
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
  warnings.warn(
model.safetensors: 100% 268M/268M [00:03<00:00, 106MB/s]
```

```
distilbert_num_parameters = model.num_parameters() / 1_000_000
print(f"'>>> DistilBERT number of parameters:
{round(distilbert_num_parameters)}M'")
print(f"'>>> BERT number of parameters: 110M'")
```

```
'>>> DistilBERT number of parameters: 67M'
'>>> BERT number of parameters: 110M'
```

```
text = "This is a great [MASK]."
```

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
```

```
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 1.99kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 659kB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 2.62MB/s]
```

```python
import torch

inputs = tokenizer(text, return_tensors="pt")
token_logits = model(**inputs).logits
# Find the location of [MASK] and extract its logits
mask_token_index = torch.where(inputs["input_ids"] ==
tokenizer.mask_token_id)[1]
mask_token_logits = token_logits[0, mask_token_index, :]
# Pick the [MASK] candidates with the highest logits
top_5_tokens = torch.topk(mask_token_logits, 5, dim=1).indices[0].tolist()

for token in top_5_tokens:
    print(f"'>>> {text.replace(tokenizer.mask_token,
tokenizer.decode([token]))}'")
```

```
'>>> This is a great deal.'
'>>> This is a great success.'
'>>> This is a great adventure.'
'>>> This is a great idea.'
'>>> This is a great feat.'
```

```python
from datasets import load_dataset

imdb_dataset = load_dataset("imdb")
imdb_dataset
```

```
README.md: 100% 7.81k/7.81k [00:00<00:00, 70.2kB/s]
train-00000-of-00001.parquet: 100% 21.0M/21.0M [00:00<00:00, 51.5MB/s]
test-00000-of-00001.parquet: 100% 20.5M/20.5M [00:00<00:00, 177MB/s]
unsupervised-00000-of-00001.parquet: 100% 42.0M/42.0M [00:00<00:00, 184MB/s]
Generating train split: 100% 25000/25000 [00:00<00:00, 49551.82 examples/s]
Generating test split: 100% 25000/25000 [00:00<00:00, 79170.57 examples/s]
Generating unsupervised split: 100% 50000/50000 [00:01<00:00, 51807.48 examples/s]
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    unsupervised: Dataset({
        features: ['text', 'label'],
        num_rows: 50000
    })
})
```

```python
sample = imdb_dataset["train"].shuffle(seed=42).select(range(3))

for row in sample:
    print(f"\n'>>> Review: {row['text']}'")
    print(f"'>>> Label: {row['label']}'")
```

```
'>>> Review: There is no relation at all between Fortier and Profiler but the fact that both are p
'>>> Label: 1'

'>>> Review: This movie is a great. The plot is very true to the book which is a classic written b
'>>> Label: 1'

'>>> Review: George P. Cosmatos' "Rambo: First Blood Part II" is pure wish-fulfillment. The United
'>>> Label: 0'
```

```python
def tokenize_function(examples):
    result = tokenizer(examples["text"])
    if tokenizer.is_fast:
        result["word_ids"] = [result.word_ids(i) for i in
range(len(result["input_ids"]))]
    return result


# Use batched=True to activate fast multithreading!
tokenized_datasets = imdb_dataset.map(
    tokenize_function, batched=True, remove_columns=["text", "label"]
)
tokenized_datasets
```

```
Map: 100% 25000/25000 [00:29<00:00, 962.77 examples/s]
Token indices sequence length is longer than the specified maximum sequence length for this model (
Map: 100% 25000/25000 [00:30<00:00, 1004.59 examples/s]
Map: 100% 50000/50000 [00:58<00:00, 888.42 examples/s]
DatasetDict({
    train: Dataset({
        features: ['input_ids', 'attention_mask', 'word_ids'],
        num_rows: 25000
    })
    test: Dataset({
        features: ['input_ids', 'attention_mask', 'word_ids'],
        num_rows: 25000
    })
    unsupervised: Dataset({
        features: ['input_ids', 'attention_mask', 'word_ids'],
        num_rows: 50000
    })
})
```

```
tokenizer.model_max_length
```

```
512
```

```
chunk_size = 128


# Slicing produces a list of lists for each feature
tokenized_samples = tokenized_datasets["train"][:3]

for idx, sample in enumerate(tokenized_samples["input_ids"]):
    print(f"'>>> Review {idx} length: {len(sample)}'")
```

```
'>>> Review 0 length: 363'
'>>> Review 1 length: 304'
'>>> Review 2 length: 133'
```

```
concatenated_examples = {
    k: sum(tokenized_samples[k], []) for k in tokenized_samples.keys()
}
total_length = len(concatenated_examples["input_ids"])
print(f"'>>> Concatenated reviews length: {total_length}'")
```

```
'>>> Concatenated reviews length: 800'
```

```
chunks = {
    k: [t[i : i + chunk_size] for i in range(0, total_length, chunk_size)]
    for k, t in concatenated_examples.items()
}

for chunk in chunks["input_ids"]:
    print(f"'>>> Chunk length: {len(chunk)}'")
```

```
'>>> Chunk length: 128'
'>>> Chunk length: 128'
'>>> Chunk length: 128'
'>>> Chunk length: 128'
'>>> Chunk length: 128'
'>>> Chunk length: 128'
'>>> Chunk length: 32'
```

```
def group_texts(examples):
    # Concatenate all texts
    concatenated_examples = {k: sum(examples[k], []) for k in
examples.keys()}
    # Compute length of concatenated texts
    total_length = len(concatenated_examples[list(examples.keys())[0]])
    # We drop the last chunk if it's smaller than chunk_size
    total_length = (total_length // chunk_size) * chunk_size
    # Split by chunks of max_len
    result = {
        k: [t[i : i + chunk_size] for i in range(0, total_length,
chunk_size)]
        for k, t in concatenated_examples.items()
    }
    # Create a new labels column
    result["labels"] = result["input_ids"].copy()
    return result
```

```
[ ] lm_datasets = tokenized_datasets.map(group_texts, batched=True)
    lm_datasets
```

```
lm_datasets = tokenized_datasets.map(group_texts, batched=True)
lm_datasets
```

```
Map: 100% 25000/25000 [01:23<00:00, 288.86 examples/s]
Map: 100% 25000/25000 [01:11<00:00, 367.46 examples/s]
Map:  64% 32000/50000 [01:34<00:54, 329.60 examples/s]
Exception ignored in: <function _xla_gc_callback at 0x793d7ef51900>
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/jax/_src/lib/__init__.py", line 96, in _xla_gc_call
    def _xla_gc_callback(*args):
KeyboardInterrupt:
```

```
tokenizer.decode(lm_datasets["train"][1]["input_ids"])
```

```
from transformers import DataCollatorForLanguageModeling

data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer,
mlm_probability=0.15)
```

```
samples = [lm_datasets["train"][i] for i in range(2)]
for sample in samples:
    _ = sample.pop("word_ids")

for chunk in data_collator(samples)["input_ids"]:
    print(f"\n'>>> {tokenizer.decode(chunk)}'")
```

```
import collections
import numpy as np

from transformers import default_data_collator

wwm_probability = 0.2


def whole_word_masking_data_collator(features):
    for feature in features:
        word_ids = feature.pop("word_ids")

        # Create a map between words and corresponding token indices
        mapping = collections.defaultdict(list)
        current_word_index = -1
        current_word = None
```

```
        for idx, word_id in enumerate(word_ids):
            if word_id is not None:
                if word_id != current_word:
                    current_word = word_id
                    current_word_index += 1
                mapping[current_word_index].append(idx)

        # Randomly mask words
        mask = np.random.binomial(1, wwm_probability, (len(mapping),))
        input_ids = feature["input_ids"]
        labels = feature["labels"]
        new_labels = [-100] * len(labels)
        for word_id in np.where(mask)[0]:
            word_id = word_id.item()
            for idx in mapping[word_id]:
                new_labels[idx] = labels[idx]
                input_ids[idx] = tokenizer.mask_token_id
        feature["labels"] = new_labels

    return default_data_collator(features)


samples = [lm_datasets["train"][i] for i in range(2)]
batch = whole_word_masking_data_collator(samples)

for chunk in batch["input_ids"]:
    print(f"\n'>>> {tokenizer.decode(chunk)}'")
```

```
'>>> [CLS] bromwell high is a cartoon comedy [MASK] it ran at the same time as some other programs about school life, such as " teachers ". my 35 years in the teaching profession lead me to believe that bromwell high\'s satire is much closer to reality than is " teachers ". the scramble to survive financially, the insightful students who can see right through their pathetic teachers\'pomp, the pettiness of the whole situation, all remind me of the schools i knew and their students. when i saw the episode in which a student repeatedly tried to burn down the school, i immediately recalled.....'

'>>> .... [MASK] [MASK] [MASK] [MASK]..... high. a classic line : inspector : i\'m here to sack one of your teachers. student : welcome to bromwell high. i expect that many adults of my age think that bromwell high is far fetched. what a pity that it isn\'t! [SEP] [CLS] homelessness ( or houselessness as george carlin stated ) has been an issue for years but never a plan to help those on the street that were once considered human who did everything from going to school, work, or vote for the matter. most people think of the homeless'
```

```
train_size = 10_000
test_size = int(0.1 * train_size)

downsampled_dataset = lm_datasets["train"].train_test_split(
    train_size=train_size, test_size=test_size, seed=42
)
downsampled_dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['attention_mask', 'input_ids', 'labels', 'word_ids'],
        num_rows: 10000
    })
    test: Dataset({
        features: ['attention_mask', 'input_ids', 'labels', 'word_ids'],
        num_rows: 1000
    })
})
```

```
from huggingface_hub import notebook_login

notebook_login()
```

```
from transformers import TrainingArguments

batch_size = 64
# Show the training loss with every epoch
logging_steps = len(downsampled_dataset["train"]) // batch_size
model_name = model_checkpoint.split("/")[-1]

training_args = TrainingArguments(
    output_dir=f"{model_name}-finetuned-imdb",
    overwrite_output_dir=True,
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    weight_decay=0.01,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    push_to_hub=True,
    fp16=True,
    logging_steps=logging_steps,
)
```

```python
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=downsampled_dataset["train"],
    eval_dataset=downsampled_dataset["test"],
    data_collator=data_collator,
    tokenizer=tokenizer,
)

import math

eval_results = trainer.evaluate()
print(f">>> Perplexity: {math.exp(eval_results['eval_loss']):.2f}")
```

```
>>> Perplexity: 21.75
```

```python
trainer.train()
```

```python
eval_results = trainer.evaluate()
print(f">>> Perplexity: {math.exp(eval_results['eval_loss']):.2f}")
```

```
>>> Perplexity: 11.32
```

```python
trainer.push_to_hub()
```

```python
def insert_random_mask(batch):
    features = [dict(zip(batch, t)) for t in zip(*batch.values())]
    masked_inputs = data_collator(features)
    # Create a new "masked" column for each column in the dataset
    return {"masked_" + k: v.numpy() for k, v in masked_inputs.items()}
```

```
downsampled_dataset = downsampled_dataset.remove_columns(["word_ids"])
eval_dataset = downsampled_dataset["test"].map(
    insert_random_mask,
    batched=True,
    remove_columns=downsampled_dataset["test"].column_names,
)
eval_dataset = eval_dataset.rename_columns(
    {
        "masked_input_ids": "input_ids",
        "masked_attention_mask": "attention_mask",
        "masked_labels": "labels",
    }
)


from torch.utils.data import DataLoader
from transformers import default_data_collator

batch_size = 64
train_dataloader = DataLoader(
    downsampled_dataset["train"],
    shuffle=True,
    batch_size=batch_size,
    collate_fn=data_collator,
)
eval_dataloader = DataLoader(
    eval_dataset, batch_size=batch_size, collate_fn=default_data_collator
)


from torch.optim import AdamW

optimizer = AdamW(model.parameters(), lr=5e-5)


from accelerate import Accelerator

accelerator = Accelerator()
model, optimizer, train_dataloader, eval_dataloader = accelerator.prepare(
    model, optimizer, train_dataloader, eval_dataloader
)
```

```
from transformers import get_scheduler

num_train_epochs = 3
num_update_steps_per_epoch = len(train_dataloader)
num_training_steps = num_train_epochs * num_update_steps_per_epoch

lr_scheduler = get_scheduler(
    "linear",
    optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=num_training_steps,
)
```

```
from huggingface_hub import get_full_repo_name

model_name = "distilbert-base-uncased-finetuned-imdb-accelerate"
repo_name = get_full_repo_name(model_name)
repo_name
```

```
'lewtun/distilbert-base-uncased-finetuned-imdb-accelerate'
```

```
from huggingface_hub import Repository

output_dir = model_name
repo = Repository(output_dir, clone_from=repo_name)
```

```
from tqdm.auto import tqdm
import torch
import math

progress_bar = tqdm(range(num_training_steps))

for epoch in range(num_train_epochs):
    # Training
    model.train()
    for batch in train_dataloader:
        outputs = model(**batch)
        loss = outputs.loss
        accelerator.backward(loss)

        optimizer.step()
        lr_scheduler.step()
```

```
        optimizer.zero_grad()
        progress_bar.update(1)

    # Evaluation
    model.eval()
    losses = []
    for step, batch in enumerate(eval_dataloader):
        with torch.no_grad():
            outputs = model(**batch)

        loss = outputs.loss
        losses.append(accelerator.gather(loss.repeat(batch_size)))

    losses = torch.cat(losses)
    losses = losses[: len(eval_dataset)]
    try:
        perplexity = math.exp(torch.mean(losses))
    except OverflowError:
        perplexity = float("inf")

    print(f">>> Epoch {epoch}: Perplexity: {perplexity}")

    # Save and upload
    accelerator.wait_for_everyone()
    unwrapped_model = accelerator.unwrap_model(model)
    unwrapped_model.save_pretrained(output_dir,
save_function=accelerator.save)
    if accelerator.is_main_process:
        tokenizer.save_pretrained(output_dir)
        repo.push_to_hub(
            commit_message=f"Training in progress epoch {epoch}",
blocking=False
        )
```

```
>>> Epoch 0: Perplexity: 11.397545307900472
>>> Epoch 1: Perplexity: 10.904909330983092
>>> Epoch 2: Perplexity: 10.729503505340409
```

```
from transformers import pipeline

mask_filler = pipeline(
    "fill-mask", model="huggingface-course/distilbert-base-uncased-
finetuned-imdb"
```

```
)

preds = mask_filler(text)

for pred in preds:
    print(f">>> {pred['sequence']}")
```

```
'>>> this is a great movie.'
'>>> this is a great film.'
'>>> this is a great story.'
'>>> this is a great movies.'
'>>> this is a great character.'
```

- **Translation**

```
[ ] !pip install datasets evaluate transformers[sentencepiece]
    !pip install accelerate
    # To run the training on TPU, you will need to uncomment the following line:
    # !pip install cloud-tpu-client==0.10 torch==1.9.0 https://storage.googleapis.com/tpu-pytorch/whee
    !apt install git-lfs
```

```
Collecting datasets
  Downloading datasets-3.2.0-py3-none-any.whl.metadata (20 kB)
Collecting evaluate
  Downloading evaluate-0.4.3-py3-none-any.whl.metadata (9.2 kB)
Requirement already satisfied: transformers[sentencepiece] in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from datasets)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from dataset
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.10/dist-packages (from da
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from datasets) (2
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.10/dist-packages (from da
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.10/dist-packages (from datase
Collecting xxhash (from datasets)
  Downloading xxhash-3.5.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12
Collecting multiprocess<0.70.17 (from datasets)
  Downloading multiprocess-0.70.16-py310-none-any.whl.metadata (7.2 kB)
Collecting fsspec<=2024.9.0,>=2023.1.0 (from fsspec[http]<=2024.9.0,>=2023.1.0->datasets)
  Downloading fsspec-2024.9.0-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from datasets)
Requirement already satisfied: huggingface-hub>=0.23.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from datasets)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from dataset
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: sentencepiece!=0.1.92,>=0.1.91 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: protobuf in /usr/local/lib/python3.10/dist-packages (from transforme
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from ai
Requirement already satisfied: async-timeout<6.0,>=4.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aioht
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from
```

```
[ ]  !git config --global user.email "you@example.com"
     !git config --global user.name "Your Name"
```

Kode ini digunakan untuk mengatur konfigurasi global Git dengan menetapkan email dan nama pengguna. Konfigurasi ini memastikan setiap kontribusi pada repositori Git memiliki identitas yang jelas, memudahkan pelacakan perubahan, dan menjaga keterhubungan dengan platform pengelolaan versi seperti GitHub atau GitLab.

```
[ ]  from huggingface_hub import notebook_login

     notebook_login()
```

Kode ini digunakan untuk autentikasi ke Hugging Face Hub melalui notebook. Dengan menjalankan notebook_login(), pengguna dapat memasukkan token akses mereka untuk mendapatkan izin mengelola model, dataset, dan sumber daya lain di Hugging Face Hub secara langsung dari lingkungan notebook, sehingga mempermudah integrasi dan kolaborasi dalam proyek NLP.

```
[ ]  from datasets import load_dataset

     raw_datasets = load_dataset("kde4", lang1="en", lang2="fr")
```

```
[ ]  raw_datasets
```

```
DatasetDict({
    train: Dataset({
        features: ['id', 'translation'],
        num_rows: 210173
    })
})
```

```
[ ] split_datasets = raw_datasets["train"].train_test_split(train_size=0.9, seed=20)
    split_datasets
```

```
DatasetDict({
    train: Dataset({
        features: ['id', 'translation'],
        num_rows: 189155
    })
    test: Dataset({
        features: ['id', 'translation'],
        num_rows: 21018
    })
})
```

```
[ ] split_datasets["validation"] = split_datasets.pop("test")
```

```
[ ] split_datasets["train"][1]["translation"]
```

```
{'en': 'Default to expanded threads',
 'fr': 'Par défaut, développer les fils de discussion'}
```

```
[ ] from transformers import pipeline

    model_checkpoint = "Helsinki-NLP/opus-mt-en-fr"
    translator = pipeline("translation", model=model_checkpoint)
    translator("Default to expanded threads")
```

```
[{'translation_text': 'Par défaut pour les threads élargis'}]
```

```
[ ] split_datasets["train"][172]["translation"]
```

```
{'en': 'Unable to import %1 using the OFX importer plugin. This file is not the correct format.',
 'fr': "Impossible d'importer %1 en utilisant le module d'extension d'importation OFX. Ce fichier
n'a pas un format correct."}
```

```
[ ] translator(
        "Unable to import %1 using the OFX importer plugin. This file is not the correct format."
    )
```

```
[{'translation_text': "Impossible d'importer %1 en utilisant le plugin d'importateur OFX. Ce
fichier n'est pas le bon format."}]
```

```
[ ] from transformers import AutoTokenizer

    model_checkpoint = "Helsinki-NLP/opus-mt-en-fr"
    tokenizer = AutoTokenizer.from_pretrained(model_checkpoint, return_tensors="pt")
```

```
[ ] en_sentence = split_datasets["train"][1]["translation"]["en"]
    fr_sentence = split_datasets["train"][1]["translation"]["fr"]

    inputs = tokenizer(en_sentence, text_target=fr_sentence)
    inputs
```

```
{'input_ids': [47591, 12, 9842, 19634, 9, 0], 'attention_mask': [1, 1, 1, 1, 1, 1], 'labels':
[577, 5891, 2, 3184, 16, 2542, 5, 1710, 0]}
```

```
[ ] wrong_targets = tokenizer(fr_sentence)
    print(tokenizer.convert_ids_to_tokens(wrong_targets["input_ids"]))
    print(tokenizer.convert_ids_to_tokens(inputs["labels"]))
```

```
['_Par', '_dé', 'f', 'aut', ',', '_dé', 've', 'lop', 'per', '_les', '_fil', 's', '_de',
'_discussion', '</s>']
['_Par', '_défaut', ',', '_développer', '_les', '_fils', '_de', '_discussion', '</s>']
```

```
[ ]  max_length = 128


     def preprocess_function(examples):
         inputs = [ex["en"] for ex in examples["translation"]]
         targets = [ex["fr"] for ex in examples["translation"]]
         model_inputs = tokenizer(
             inputs, text_target=targets, max_length=max_length, truncation=True
         )
         return model_inputs
```

```
[ ]  tokenized_datasets = split_datasets.map(
         preprocess_function,
         batched=True,
         remove_columns=split_datasets["train"].column_names,
     )
```

```
[ ]  from transformers import AutoModelForSeq2SeqLM

     model = AutoModelForSeq2SeqLM.from_pretrained(model_checkpoint)
```

```
[ ]  from transformers import DataCollatorForSeq2Seq

     data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)
```

```
[ ]  batch = data_collator([tokenized_datasets["train"][i] for i in range(1, 3)])
     batch.keys()
```

```
dict_keys(['attention_mask', 'input_ids', 'labels', 'decoder_input_ids'])
```

```
[ ] batch["labels"]
```

```
tensor([[  577,  5891,     2,  3184,    16,  2542,     5,  1710,     0,  -100,
           -100,  -100,  -100,  -100,  -100,  -100],
        [ 1211,     3,    49,  9409,  1211,     3, 29140,   817,  3124,   817,
            550,  7032,  5821,  7907, 12649,     0]])
```

```
[ ] batch["decoder_input_ids"]
```

```
tensor([[59513,   577,  5891,     2,  3184,    16,  2542,     5,  1710,     0,
         59513, 59513, 59513, 59513, 59513, 59513],
        [59513,  1211,     3,    49,  9409,  1211,     3, 29140,   817,  3124,
           817,   550,  7032,  5821,  7907, 12649]])
```

```
[ ] for i in range(1, 3):
        print(tokenized_datasets["train"][i]["labels"])
```

```
[577, 5891, 2, 3184, 16, 2542, 5, 1710, 0]
[1211, 3, 49, 9409, 1211, 3, 29140, 817, 3124, 817, 550, 7032, 5821, 7907, 12649, 0]
```

```
[ ] !pip install sacrebleu
```

```
[ ] import evaluate

    metric = evaluate.load("sacrebleu")
```

```
[ ] predictions = [
        "This plugin lets you translate web pages between several languages automatically."
    ]
    references = [
        [
            "This plugin allows you to automatically translate web pages between several languages."
        ]
    ]
    metric.compute(predictions=predictions, references=references)
```

```
{'score': 46.750469682990165,
 'counts': [11, 6, 4, 3],
 'totals': [12, 11, 10, 9],
 'precisions': [91.67, 54.54, 40.0, 33.33],
 'bp': 0.9200444146293233,
 'sys_len': 12,
 'ref_len': 13}
```

```
[ ] predictions = ["This This This This"]
    references = [
        [
            "This plugin allows you to automatically translate web pages between several languages."
        ]
    ]
    metric.compute(predictions=predictions, references=references)
```

```
{'score': 1.683602693167689,
 'counts': [1, 0, 0, 0],
 'totals': [4, 3, 2, 1],
 'precisions': [25.0, 16.67, 12.5, 12.5],
 'bp': 0.10539922456186433,
 'sys_len': 4,
 'ref_len': 13}
```

```
[ ] predictions = ["This plugin"]
    references = [
        [
            "This plugin allows you to automatically translate web pages between several languages."
        ]
    ]
    metric.compute(predictions=predictions, references=references)
```

```
{'score': 0.0,
 'counts': [2, 1, 0, 0],
 'totals': [2, 1, 0, 0],
 'precisions': [100.0, 100.0, 0.0, 0.0],
 'bp': 0.004086771438464067,
 'sys_len': 2,
 'ref_len': 13}
```

```
[ ] import numpy as np


    def compute_metrics(eval_preds):
        preds, labels = eval_preds
        # In case the model returns more than the prediction logits
        if isinstance(preds, tuple):
            preds = preds[0]

        decoded_preds = tokenizer.batch_decode(preds, skip_special_tokens=True)

        # Replace -100s in the labels as we can't decode them
        labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
        decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)

        # Some simple post-processing
        decoded_preds = [pred.strip() for pred in decoded_preds]
        decoded_labels = [[label.strip()] for label in decoded_labels]
```

```
[ ]  from huggingface_hub import notebook_login

     notebook_login()
```

```
[ ]  from transformers import Seq2SeqTrainingArguments

     args = Seq2SeqTrainingArguments(
         f"marian-finetuned-kde4-en-to-fr",
         evaluation_strategy="no",
         save_strategy="epoch",
         learning_rate=2e-5,
         per_device_train_batch_size=32,
         per_device_eval_batch_size=64,
         weight_decay=0.01,
         save_total_limit=3,
         num_train_epochs=3,
         predict_with_generate=True,
         fp16=True,
         push_to_hub=True,
     )
```

```
[ ]  from transformers import Seq2SeqTrainer

     trainer = Seq2SeqTrainer(
         model,
         args,
         train_dataset=tokenized_datasets["train"],
         eval_dataset=tokenized_datasets["validation"],
         data_collator=data_collator,
         tokenizer=tokenizer,
         compute_metrics=compute_metrics,
     )
```

```
[ ] trainer.evaluate(max_length=max_length)
```

```
{'eval_loss': 1.6964408159255981,
 'eval_bleu': 39.26865061007616,
 'eval_runtime': 965.8884,
 'eval_samples_per_second': 21.76,
 'eval_steps_per_second': 0.341}
```

```
[ ] trainer.train()
```

```
[ ] trainer.evaluate(max_length=max_length)
```

```
{'eval_loss': 0.8558505773544312,
 'eval_bleu': 52.94161337775576,
 'eval_runtime': 714.2576,
 'eval_samples_per_second': 29.426,
 'eval_steps_per_second': 0.461,
 'epoch': 3.0}
```

```
[ ] trainer.push_to_hub(tags="translation", commit_message="Training complete")
```

```
'https://huggingface.co/sgugger/marian-finetuned-kde4-en-to-fr/commit/3601d621e3baae2bc63d3311452535f8f58f6ef3'
```

```
[ ] from torch.utils.data import DataLoader

    tokenized_datasets.set_format("torch")
    train_dataloader = DataLoader(
        tokenized_datasets["train"],
        shuffle=True,
        collate_fn=data_collator,
        batch_size=8,
    )
    eval_dataloader = DataLoader(
```

```
[ ]  model = AutoModelForSeq2SeqLM.from_pretrained(model_checkpoint)
```

```
[ ]  from transformers import AdamW

     optimizer = AdamW(model.parameters(), lr=2e-5)
```

```
[ ]  from accelerate import Accelerator

     accelerator = Accelerator()
     model, optimizer, train_dataloader, eval_dataloader = accelerator.prepare(
         model, optimizer, train_dataloader, eval_dataloader
     )
```

```
[ ]  from transformers import get_scheduler

     num_train_epochs = 3
     num_update_steps_per_epoch = len(train_dataloader)
     num_training_steps = num_train_epochs * num_update_steps_per_epoch

     lr_scheduler = get_scheduler(
         "linear",
         optimizer=optimizer,
         num_warmup_steps=0,
         num_training_steps=num_training_steps,
     )
```

```
[ ]  from huggingface_hub import Repository, get_full_repo_name

     model_name = "marian-finetuned-kde4-en-to-fr-accelerate"
     repo_name = get_full_repo_name(model_name)
     repo_name
```

```
'sgugger/marian-finetuned-kde4-en-to-fr-accelerate'
```

```
[ ]  output_dir = "marian-finetuned-kde4-en-to-fr-accelerate"
     repo = Repository(output_dir, clone_from=repo_name)
```

```
[ ]  def postprocess(predictions, labels):
         predictions = predictions.cpu().numpy()
         labels = labels.cpu().numpy()

         decoded_preds = tokenizer.batch_decode(predictions, skip_special_tokens=True)

         # Replace -100 in the labels as we can't decode them.
         labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
         decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)

         # Some simple post-processing
         decoded_preds = [pred.strip() for pred in decoded_preds]
         decoded_labels = [[label.strip()] for label in decoded_labels]
         return decoded_preds, decoded_labels
```

```
[ ]  from tqdm.auto import tqdm
     import torch

     progress_bar = tqdm(range(num_training_steps))

     for epoch in range(num_train_epochs):
         # Training
         model.train()
         for batch in train_dataloader:
             outputs = model(**batch)
             loss = outputs.loss
             accelerator.backward(loss)

             optimizer.step()
             lr_scheduler.step()
             optimizer.zero_grad()
             progress_bar.update(1)
```

```
[ ]         with torch.no_grad():
                generated_tokens = accelerator.unwrap_model(model).generate(
                    batch["input_ids"],
                    attention_mask=batch["attention_mask"],
                    max_length=128,
                )
            labels = batch["labels"]

            # Necessary to pad predictions and labels for being gathered
            generated_tokens = accelerator.pad_across_processes(
                generated_tokens, dim=1, pad_index=tokenizer.pad_token_id
            )
            labels = accelerator.pad_across_processes(labels, dim=1, pad_index=-100)

            predictions_gathered = accelerator.gather(generated_tokens)
            labels_gathered = accelerator.gather(labels)

            decoded_preds, decoded_labels = postprocess(predictions_gathered, labels_gathered)
            metric.add_batch(predictions=decoded_preds, references=decoded_labels)

        results = metric.compute()
        print(f"epoch {epoch}, BLEU score: {results['score']:.2f}")

        # Save and upload
        accelerator.wait_for_everyone()
        unwrapped_model = accelerator.unwrap_model(model)
        unwrapped_model.save_pretrained(output_dir, save_function=accelerator.save)
        if accelerator.is_main_process:
            tokenizer.save_pretrained(output_dir)
            repo.push_to_hub(
                commit_message=f"Training in progress epoch {epoch}", blocking=False
            )
```

```
epoch 0, BLEU score: 53.47
epoch 1, BLEU score: 54.24
epoch 2, BLEU score: 54.44
```

```
[ ] from transformers import pipeline

    # Replace this with your own checkpoint
    model_checkpoint = "huggingface-course/marian-finetuned-kde4-en-to-fr"
    translator = pipeline("translation", model=model_checkpoint)
    translator("Default to expanded threads")
```

```
[{'translation_text': 'Par défaut, développer les fils de discussion'}]
```

```
[ ] translator(
        "Unable to import %1 using the OFX importer plugin. This file is not the correct format."
    )
```

```
[{'translation_text': "Impossible d'importer %1 en utilisant le module externe d'importation OFX.
Ce fichier n'est pas le bon format."}]
```

- Summarization

```
[ ] !pip install datasets evaluate transformers[sentencepiece]
    !pip install accelerate
    # To run the training on TPU, you will need to uncomment the following line:
    # !pip install cloud-tpu-client==0.10 torch==1.9.0 https://storage.googleapis.com/tpu-pytorch/whe
    !apt install git-lfs
```

```
Collecting datasets
  Downloading datasets-3.2.0-py3-none-any.whl.metadata (20 kB)
Collecting evaluate
  Downloading evaluate-0.4.3-py3-none-any.whl.metadata (9.2 kB)
Requirement already satisfied: transformers[sentencepiece] in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from datasets)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from datase
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.10/dist-packages (from da
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from datasets) (
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.10/dist-packages (from d
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.10/dist-packages (from datas
Collecting xxhash (from datasets)
  Downloading xxhash-3.5.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12
Collecting multiprocess<0.70.17 (from datasets)
  Downloading multiprocess-0.70.16-py310-none-any.whl.metadata (7.2 kB)
Collecting fsspec<=2024.9.0,>=2023.1.0 (from fsspec[http]<=2024.9.0,>=2023.1.0->datasets)
  Downloading fsspec-2024.9.0-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from datasets)
Requirement already satisfied: huggingface-hub>=0.23.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from datasets
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from datase
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from
```

```
[ ] !git config --global user.email "you@example.com"
    !git config --global user.name "Your Name"
```

Kode ini digunakan untuk mengatur konfigurasi global Git dengan menetapkan email dan nama pengguna. Konfigurasi ini memastikan setiap kontribusi pada repositori Git memiliki identitas yang jelas, memudahkan pelacakan perubahan, dan menjaga keterhubungan dengan platform pengelolaan versi seperti GitHub atau GitLab.

```
[ ] from huggingface_hub import notebook_login

    notebook_login()
```

Kode ini digunakan untuk autentikasi ke Hugging Face Hub melalui notebook. Dengan menjalankan notebook_login(), pengguna dapat memasukkan token akses mereka untuk mendapatkan izin mengelola model, dataset, dan sumber daya lain di Hugging Face Hub secara langsung dari lingkungan notebook, sehingga mempermudah integrasi dan kolaborasi dalam proyek NLP.

```
[ ] from datasets import load_dataset

    spanish_dataset = load_dataset("amazon_reviews_multi", "es")
    english_dataset = load_dataset("amazon_reviews_multi", "en")
    english_dataset
```

```
[ ] from datasets import load_dataset

    spanish_dataset = load_dataset("amazon_reviews_multi", "es")
    english_dataset = load_dataset("amazon_reviews_multi", "en")
    english_dataset
```

```
[ ] def show_samples(dataset, num_samples=3, seed=42):
        sample = dataset["train"].shuffle(seed=seed).select(range(num_samples))
        for example in sample:
            print(f"\n'>> Title: {example['review_title']}'")
            print(f"'>> Review: {example['review_body']}'")


    show_samples(english_dataset)
```

```
'>> Title: Worked in front position, not rear'
'>> Review: 3 stars because these are not rear brakes as stated in the item description. At least
the mount adapter only worked on the front fork of the bike that I got it for.'

'>> Title: meh'
'>> Review: Does it’s job and it’s gorgeous but mine is falling apart, I had to basically put it
together again with hot glue'

'>> Title: Can\'t beat these for the money'
'>> Review: Bought this for handling miscellaneous aircraft parts and hanger "stuff" that I needed
to organize; it really fit the bill. The unit arrived quickly, was well packaged and arrived
intact (always a good sign). There are five wall mounts-- three on the top and two on the bottom.
I wanted to mount it on the wall, so all I had to do was to remove the top two layers of plastic
drawers, as well as the bottom corner drawers, place it when I wanted and mark it; I then used
some of the new plastic screw in wall anchors (the 50 pound variety) and it easily mounted to the
wall. Some have remarked that they wanted dividers for the drawers, and that they made those. Good
idea. My application was that I needed something that I can see the contents at about eye level,
so I wanted the fuller-sized drawers. I also like that these are the new plastic that doesn\'t get
brittle and split like my older plastic drawers did. I like the all-plastic construction. It\'s
heavy duty enough to hold metal parts, but being made of plastic it\'s not as heavy as a metal
frame, so you can easily mount it to the wall and still load it up with heavy stuff, or light
stuff. No problem there. For the money, you can\'t beat it. Best one of these I\'ve bought to
```

```
[ ]  english_dataset.set_format("pandas")
     english_df = english_dataset["train"][:]
     # Show counts for top 20 products
     english_df["product_category"].value_counts()[:20]
```

```
home                      17679
apparel                   15951
wireless                  15717
other                     13418
beauty                    12091
drugstore                 11730
kitchen                   10382
toy                        8745
sports                     8277
automotive                 7506
lawn_and_garden            7327
home_improvement           7136
pet_products               7082
digital_ebook_purchase     6749
pc                         6401
electronics                6186
office_product             5521
shoes                      5197
grocery                    4730
book                       3756
Name: product_category, dtype: int64
```

```
[ ]  def filter_books(example):
         return (
             example["product_category"] == "book"
             or example["product_category"] == "digital_ebook_purchase"
         )
```

```
[ ]  english_dataset.reset_format()
```

```
[ ] spanish_books = spanish_dataset.filter(filter_books)
    english_books = english_dataset.filter(filter_books)
    show_samples(english_books)
```

```
'>> Title: I\'m dissapointed.'
'>> Review: I guess I had higher expectations for this book from the reviews. I really thought
I\'d at least like it. The plot idea was great. I loved Ash but, it just didnt go anywhere. Most
of the book was about their radio show and talking to callers. I wanted the author to dig deeper
so we could really get to know the characters. All we know about Grace is that she is attractive
looking, Latino and is kind of a brat. I\'m dissapointed.'

'>> Title: Good art, good price, poor design'
'>> Review: I had gotten the DC Vintage calendar the past two years, but it was on backorder
forever this year and I saw they had shrunk the dimensions for no good reason. This one has good
art choices but the design has the fold going through the picture, so it\'s less aesthetically
pleasing, especially if you want to keep a picture to hang. For the price, a good calendar'

'>> Title: Helpful'
'>> Review: Nearly all the tips useful and. I consider myself an intermediate to advanced user of
OneNote. I would highly recommend.'
```

```
[ ] from datasets import concatenate_datasets, DatasetDict

    books_dataset = DatasetDict()

    for split in english_books.keys():
        books_dataset[split] = concatenate_datasets(
            [english_books[split], spanish_books[split]]
        )
        books_dataset[split] = books_dataset[split].shuffle(seed=42)

    # Peek at a few examples
    show_samples(books_dataset)
```

```
'>> Title: Easy to follow!!!!'
'>> Review: I loved The dash diet weight loss Solution. Never hungry. I would recommend this diet.
Also the menus are well rounded. Try it. Has lots of the information need thanks.'
```

```
[ ] books_dataset = books_dataset.filter(lambda x: len(x["review_title"].split()) > 2)
```

```
[ ] from transformers import AutoTokenizer

    model_checkpoint = "google/mt5-small"
    tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
```

```
[ ] inputs = tokenizer("I loved reading the Hunger Games!")
    inputs
```

```
{'input_ids': [336, 259, 28387, 11807, 287, 62893, 295, 12507, 1], 'attention_mask': [1, 1, 1, 1,
1, 1, 1, 1, 1]}
```

```
[ ] tokenizer.convert_ids_to_tokens(inputs.input_ids)
```

```
['_I', '_', 'loved', '_reading', '_the', '_Hung', 'er', '_Games', '</s>']
```

```
[ ] max_input_length = 512
    max_target_length = 30


    def preprocess_function(examples):
        model_inputs = tokenizer(
            examples["review_body"],
            max_length=max_input_length,
            truncation=True,
        )
        labels = tokenizer(
            examples["review_title"], max_length=max_target_length, truncation=True
        )
        model_inputs["labels"] = labels["input_ids"]
        return model_inputs
```

```
[ ] tokenized_datasets = books_dataset.map(preprocess_function, batched=True)
```

```
[ ] generated_summary = "I absolutely loved reading the Hunger Games"
    reference_summary = "I loved reading the Hunger Games"
```

```
[ ] !pip install rouge_score
```

```
[ ] import evaluate

    rouge_score = evaluate.load("rouge")
```

```
[ ] scores = rouge_score.compute(
        predictions=[generated_summary], references=[reference_summary]
    )
    scores
```

```
{'rouge1': AggregateScore(low=Score(precision=0.86, recall=1.0, fmeasure=0.92),
mid=Score(precision=0.86, recall=1.0, fmeasure=0.92), high=Score(precision=0.86, recall=1.0,
fmeasure=0.92)),
 'rouge2': AggregateScore(low=Score(precision=0.67, recall=0.8, fmeasure=0.73),
mid=Score(precision=0.67, recall=0.8, fmeasure=0.73), high=Score(precision=0.67, recall=0.8,
fmeasure=0.73)),
 'rougeL': AggregateScore(low=Score(precision=0.86, recall=1.0, fmeasure=0.92),
mid=Score(precision=0.86, recall=1.0, fmeasure=0.92), high=Score(precision=0.86, recall=1.0,
fmeasure=0.92)),
 'rougeLsum': AggregateScore(low=Score(precision=0.86, recall=1.0, fmeasure=0.92),
mid=Score(precision=0.86, recall=1.0, fmeasure=0.92), high=Score(precision=0.86, recall=1.0,
fmeasure=0.92))}
```

```
[ ] scores["rouge1"].mid
```

```
Score(precision=0.86, recall=1.0, fmeasure=0.92)
```

```
[ ] !pip install nltk
```

```
[ ] import nltk

    nltk.download("punkt")
```

```
[ ] from nltk.tokenize import sent_tokenize


    def three_sentence_summary(text):
        return "\n".join(sent_tokenize(text)[:3])


    print(three_sentence_summary(books_dataset["train"][1]["review_body"]))
```

```
'I grew up reading Koontz, and years ago, I stopped,convinced i had "outgrown" him.'
'Still,when a friend was looking for something suspenseful too read, I suggested Koontz.'
'She found Strangers.'
```

```
[ ] def evaluate_baseline(dataset, metric):
        summaries = [three_sentence_summary(text) for text in dataset["review_body"]]
        return metric.compute(predictions=summaries, references=dataset["review_title"])
```

```
[ ] import pandas as pd

    score = evaluate_baseline(books_dataset["validation"], rouge_score)
    rouge_names = ["rouge1", "rouge2", "rougeL", "rougeLsum"]
    rouge_dict = dict((rn, round(score[rn].mid.fmeasure * 100, 2)) for rn in rouge_names)
    rouge_dict
```

```
{'rouge1': 16.74, 'rouge2': 8.83, 'rougeL': 15.6, 'rougeLsum': 15.96}
```

```
[ ]  from transformers import AutoModelForSeq2SeqLM

     model = AutoModelForSeq2SeqLM.from_pretrained(model_checkpoint)
```

```
[ ]  from huggingface_hub import notebook_login

     notebook_login()
```

```
[ ]  from transformers import Seq2SeqTrainingArguments

     batch_size = 8
     num_train_epochs = 8
     # Show the training loss with every epoch
     logging_steps = len(tokenized_datasets["train"]) // batch_size
     model_name = model_checkpoint.split("/")[-1]

     args = Seq2SeqTrainingArguments(
         output_dir=f"{model_name}-finetuned-amazon-en-es",
         evaluation_strategy="epoch",
         learning_rate=5.6e-5,
         per_device_train_batch_size=batch_size,
         per_device_eval_batch_size=batch_size,
         weight_decay=0.01,
         save_total_limit=3,
         num_train_epochs=num_train_epochs,
         predict_with_generate=True,
         logging_steps=logging_steps,
         push_to_hub=True,
     )
```

```
[ ]  import numpy as np


     def compute_metrics(eval_pred):
         predictions, labels = eval_pred
         # Decode generated summaries into text
         decoded_preds = tokenizer.batch_decode(predictions, skip_special_tokens=True)
         # Replace -100 in the labels as we can't decode them
         labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
         # Decode reference summaries into text
         decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)
         # ROUGE expects a newline after each sentence
         decoded_preds = ["\n".join(sent_tokenize(pred.strip())) for pred in decoded_preds]
         decoded_labels = ["\n".join(sent_tokenize(label.strip())) for label in decoded_labels]
         # Compute ROUGE scores
         result = rouge_score.compute(
             predictions=decoded_preds, references=decoded_labels, use_stemmer=True
         )
         # Extract the median scores
         result = {key: value.mid.fmeasure * 100 for key, value in result.items()}
         return {k: round(v, 4) for k, v in result.items()}
```

```
[ ]  from transformers import DataCollatorForSeq2Seq

     data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)
```

```
[ ]  tokenized_datasets = tokenized_datasets.remove_columns(
         books_dataset["train"].column_names
     )
```

```
[ ] features = [tokenized_datasets["train"][i] for i in range(2)]
    data_collator(features)
```

```
{'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]), 'input_ids': tensor([[  1494,    259,   8622,    390,
259,    262,   2316,   3435,    955,
            772,    281,    772,   1617,    263,    305,  14701,    260,   1385,
           3031,    259,  24146,    332,   1037,    259,  43906,    305,    336,
            260,      1,      0,      0,      0,      0,      0,      0],
       [   259,  27531,  13483,    259,   7505,    260, 112240,  15192,    305,
          53198,    276,    259,  74060,    263,    260,    459,  25640,    776,
           2119,    336,    259,   2220,    259,  18896,    288,   4906,    288,
           1037,   3931,    260,   7083, 101476,   1143,    260,      1]]), 'labels': tensor([[
7483,   259,  2364, 15695,     1,  -100],
       [  259, 27531, 13483,   259,  7505,     1]]), 'decoder_input_ids': tensor([[    0,  7483,
259,  2364, 15695,     1],
       [    0,   259, 27531, 13483,   259,  7505]])}
```

```
[ ] from transformers import Seq2SeqTrainer

    trainer = Seq2SeqTrainer(
        model,
        args,
        train_dataset=tokenized_datasets["train"],
        eval_dataset=tokenized_datasets["validation"],
        data_collator=data_collator,
        tokenizer=tokenizer,
        compute_metrics=compute_metrics,
    )
```

```
[ ] trainer.train()
```

```
[ ] trainer.evaluate()
```

```
{'eval_loss': 3.028524398803711,
 'eval_rouge1': 16.9728,
 'eval_rouge2': 8.2969,
 'eval_rougeL': 16.8366,
 'eval_rougeLsum': 16.851,
 'eval_gen_len': 10.1597,
 'eval_runtime': 6.1054,
 'eval_samples_per_second': 38.982,
 'eval_steps_per_second': 4.914}
```

```
[ ] tokenized_datasets.set_format("torch")
```

```
[ ] model = AutoModelForSeq2SeqLM.from_pretrained(model_checkpoint)
```

```
[ ] from torch.utils.data import DataLoader

    batch_size = 8
    train_dataloader = DataLoader(
        tokenized_datasets["train"],
        shuffle=True,
        collate_fn=data_collator,
        batch_size=batch_size,
    )
    eval_dataloader = DataLoader(
        tokenized_datasets["validation"], collate_fn=data_collator, batch_size=batch_size
    )
```

```
[ ] from torch.optim import AdamW

    optimizer = AdamW(model.parameters(), lr=2e-5)
```

```
[ ]  from accelerate import Accelerator

     accelerator = Accelerator()
     model, optimizer, train_dataloader, eval_dataloader = accelerator.prepare(
         model, optimizer, train_dataloader, eval_dataloader
     )
```

```
[ ]  from transformers import get_scheduler

     num_train_epochs = 10
     num_update_steps_per_epoch = len(train_dataloader)
     num_training_steps = num_train_epochs * num_update_steps_per_epoch

     lr_scheduler = get_scheduler(
         "linear",
         optimizer=optimizer,
         num_warmup_steps=0,
         num_training_steps=num_training_steps,
     )
```

```
[ ]  def postprocess_text(preds, labels):
         preds = [pred.strip() for pred in preds]
         labels = [label.strip() for label in labels]

         # ROUGE expects a newline after each sentence
         preds = ["\n".join(nltk.sent_tokenize(pred)) for pred in preds]
         labels = ["\n".join(nltk.sent_tokenize(label)) for label in labels]

         return preds, labels
```

```
[ ] from huggingface_hub import get_full_repo_name

    model_name = "test-bert-finetuned-squad-accelerate"
    repo_name = get_full_repo_name(model_name)
    repo_name
```

```
'lewtun/mt5-finetuned-amazon-en-es-accelerate'
```

```
[ ] from huggingface_hub import Repository

    output_dir = "results-mt5-finetuned-squad-accelerate"
    repo = Repository(output_dir, clone_from=repo_name)
```

```
[ ] from tqdm.auto import tqdm
    import torch
    import numpy as np

    progress_bar = tqdm(range(num_training_steps))

    for epoch in range(num_train_epochs):
        # Training
        model.train()
        for step, batch in enumerate(train_dataloader):
            outputs = model(**batch)
            loss = outputs.loss
            accelerator.backward(loss)

            optimizer.step()
            lr_scheduler.step()
            optimizer.zero_grad()
            progress_bar.update(1)

        # Evaluation
        model.eval()
        for step, batch in enumerate(eval_dataloader):
```

```
[ ]     for step, batch in enumerate(eval_dataloader):
            with torch.no_grad():
                generated_tokens = accelerator.unwrap_model(model).generate(
                    batch["input_ids"],
                    attention_mask=batch["attention_mask"],
                )

                generated_tokens = accelerator.pad_across_processes(
                    generated_tokens, dim=1, pad_index=tokenizer.pad_token_id
                )
                labels = batch["labels"]

                # If we did not pad to max length, we need to pad the labels too
                labels = accelerator.pad_across_processes(
                    batch["labels"], dim=1, pad_index=tokenizer.pad_token_id
                )

                generated_tokens = accelerator.gather(generated_tokens).cpu().numpy()
                labels = accelerator.gather(labels).cpu().numpy()

                # Replace -100 in the labels as we can't decode them
                labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
                if isinstance(generated_tokens, tuple):
                    generated_tokens = generated_tokens[0]
                decoded_preds = tokenizer.batch_decode(
                    generated_tokens, skip_special_tokens=True
                )
                decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)

                decoded_preds, decoded_labels = postprocess_text(
                    decoded_preds, decoded_labels
                )

                rouge_score.add_batch(predictions=decoded_preds, references=decoded_labels)

        # Compute metrics
```

```
[ ]
```

```python
    # Compute metrics
    result = rouge_score.compute()
    # Extract the median ROUGE scores
    result = {key: value.mid.fmeasure * 100 for key, value in result.items()}
    result = {k: round(v, 4) for k, v in result.items()}
    print(f"Epoch {epoch}:", result)

    # Save and upload
    accelerator.wait_for_everyone()
    unwrapped_model = accelerator.unwrap_model(model)
    unwrapped_model.save_pretrained(output_dir, save_function=accelerator.save)
    if accelerator.is_main_process:
        tokenizer.save_pretrained(output_dir)
        repo.push_to_hub(
            commit_message=f"Training in progress epoch {epoch}", blocking=False
        )
```

```
Epoch 0: {'rouge1': 5.6351, 'rouge2': 1.1625, 'rougeL': 5.4866, 'rougeLsum': 5.5005}
Epoch 1: {'rouge1': 9.8646, 'rouge2': 3.4106, 'rougeL': 9.9439, 'rougeLsum': 9.9306}
Epoch 2: {'rouge1': 11.0872, 'rouge2': 3.3273, 'rougeL': 11.0508, 'rougeLsum': 10.9468}
Epoch 3: {'rouge1': 11.8587, 'rouge2': 4.8167, 'rougeL': 11.7986, 'rougeLsum': 11.7518}
Epoch 4: {'rouge1': 12.9842, 'rouge2': 5.5887, 'rougeL': 12.7546, 'rougeLsum': 12.7029}
Epoch 5: {'rouge1': 13.4628, 'rouge2': 6.4598, 'rougeL': 13.312, 'rougeLsum': 13.2913}
Epoch 6: {'rouge1': 12.9131, 'rouge2': 5.8914, 'rougeL': 12.6896, 'rougeLsum': 12.5701}
Epoch 7: {'rouge1': 13.3079, 'rouge2': 6.2994, 'rougeL': 13.1536, 'rougeLsum': 13.1194}
Epoch 8: {'rouge1': 13.96, 'rouge2': 6.5998, 'rougeL': 13.9123, 'rougeLsum': 13.7744}
Epoch 9: {'rouge1': 14.1192, 'rouge2': 7.0059, 'rougeL': 14.1172, 'rougeLsum': 13.9509}
```

```
[ ]  from transformers import pipeline

     hub_model_id = "huggingface-course/mt5-small-finetuned-amazon-en-es"
     summarizer = pipeline("summarization", model=hub_model_id)
```

```
[ ]  def print_summary(idx):
         review = books_dataset["test"][idx]["review_body"]
         title = books_dataset["test"][idx]["review_title"]
         summary = summarizer(books_dataset["test"][idx]["review_body"])[0]["summary_text"]
         print(f"'>>> Review: {review}'")
         print(f"\n'>>> Title: {title}'")
         print(f"\n'>>> Summary: {summary}'")
```

```
[ ]  print_summary(100)
```

```
'>>> Review: Nothing special at all about this product... the book is too small and stiff and hard
to write in. The huge sticker on the back doesn’t come off and looks super tacky. I would not
purchase this again. I could have just bought a journal from the dollar store and it would be
basically the same thing. It’s also really expensive for what it is.'

'>>> Title: Not impressed at all... buy something else'

'>>> Summary: Nothing special at all about this product'
```

```
[ ]  print_summary(0)
```

```
'>>> Review: Es una trilogia que se hace muy facil de leer. Me ha gustado, no me esperaba el final
para nada'

'>>> Title: Buena literatura para adolescentes'

'>>> Summary: Muy facil de leer'
```

- Training a causal language model from scratch

```
[ ]  !pip install datasets evaluate transformers[sentencepiece]
     !pip install accelerate
     # To run the training on TPU, you will need to uncomment the following line:
     # !pip install cloud-tpu-client==0.10 torch==1.9.0 https://storage.googleapis.com/tpu-pytorch/whe
     !apt install git-lfs
```

code ini saya gunakan agar dapat merunning codenya. Kode ini menginstal library dan alat penting untuk proyek NLP (Natural Language Processing) menggunakan Hugging Face. Library yang diinstal mencakup datasets untuk memuat dan memproses dataset, evaluate untuk mengevaluasi performa model, dan transformers untuk bekerja dengan model NLP canggih. accelerate digunakan untuk mempercepat pelatihan model, baik pada GPU maupun TPU. Selain itu, Git LFS diinstal untuk mengelola file besar seperti model atau dataset saat menggunakan Hugging Face Hub. Kode ini juga menyediakan opsi tambahan untuk mengonfigurasi pelatihan pada TPU.

```
[ ]  !git config --global user.email "you@example.com"
     !git config --global user.name "Your Name"
```

Kode ini digunakan untuk mengatur konfigurasi global Git dengan menetapkan email dan nama pengguna. Konfigurasi ini memastikan setiap kontribusi pada repositori Git memiliki identitas yang jelas, memudahkan pelacakan perubahan, dan menjaga keterhubungan dengan platform pengelolaan versi seperti GitHub atau GitLab.

```
[ ]  from huggingface_hub import notebook_login

     notebook_login()
```

Kode ini digunakan untuk autentikasi ke Hugging Face Hub melalui notebook. Dengan menjalankan notebook_login(), pengguna dapat memasukkan token akses mereka untuk mendapatkan izin mengelola model, dataset, dan sumber daya lain di Hugging Face Hub secara langsung dari lingkungan notebook, sehingga mempermudah integrasi dan kolaborasi dalam proyek NLP.

```
[ ] def any_keyword_in_string(string, keywords):
        for keyword in keywords:
            if keyword in string:
                return True
        return False
```

```
[ ] filters = ["pandas", "sklearn", "matplotlib", "seaborn"]
    example_1 = "import numpy as np"
    example_2 = "import pandas as pd"

    print(
        any_keyword_in_string(example_1, filters), any_keyword_in_string(example_2, filters)
    )
```

```
False True
```

```
[ ] from collections import defaultdict
    from tqdm import tqdm
    from datasets import Dataset


    def filter_streaming_dataset(dataset, filters):
        filtered_dict = defaultdict(list)
        total = 0
        for sample in tqdm(iter(dataset)):
            total += 1
            if any_keyword_in_string(sample["content"], filters):
                for k, v in sample.items():
                    filtered_dict[k].append(v)
        print(f"{len(filtered_dict['content'])/total:.2%} of data after filtering.")
        return Dataset.from_dict(filtered_dict)
```

```
[ ]  # This cell will take a very long time to execute, so you should skip it and go to
     # the next one!
     from datasets import load_dataset

     split = "train"  # "valid"
     filters = ["pandas", "sklearn", "matplotlib", "seaborn"]

     data = load_dataset(f"transformersbook/codeparrot-{split}", split=split, streaming=True)
     filtered_data = filter_streaming_dataset(data, filters)
```

```
3.26% of data after filtering.
```

```
[ ]  from datasets import load_dataset, DatasetDict

     ds_train = load_dataset("huggingface-course/codeparrot-ds-train", split="train")
     ds_valid = load_dataset("huggingface-course/codeparrot-ds-valid", split="validation")

     raw_datasets = DatasetDict(
         {
             "train": ds_train,  # .shuffle().select(range(50000)),
             "valid": ds_valid,  # .shuffle().select(range(500))
         }
     )

     raw_datasets
```

```
DatasetDict({
    train: Dataset({
        features: ['repo_name', 'path', 'copies', 'size', 'content', 'license'],
        num_rows: 606720
    })
    valid: Dataset({
        features: ['repo_name', 'path', 'copies', 'size', 'content', 'license'],
        num_rows: 3322
    })
})
```

```
[ ]  for key in raw_datasets["train"][0]:
         print(f"{key.upper()}: {raw_datasets['train'][0][key][:200]}")
```

```
'REPO_NAME: kmike/scikit-learn'
'PATH: sklearn/utils/__init__.py'
'COPIES: 3'
'SIZE: 10094'
'''CONTENT: """
The :mod:`sklearn.utils` module includes various utilites.
"""

from collections import Sequence

import numpy as np
from scipy.sparse import issparse
import warnings

from .murmurhash import murm
LICENSE: bsd-3-clause'''
```

```
[ ]  from transformers import AutoTokenizer

     context_length = 128
     tokenizer = AutoTokenizer.from_pretrained("huggingface-course/code-search-net-tokenizer")

     outputs = tokenizer(
         raw_datasets["train"][:2]["content"],
         truncation=True,
         max_length=context_length,
         return_overflowing_tokens=True,
         return_length=True,
     )

     print(f"Input IDs length: {len(outputs['input_ids'])}")
     print(f"Input chunk lengths: {(outputs['length'])}")
     print(f"Chunk mapping: {outputs['overflow_to_sample_mapping']}")
```

```
Input IDs length: 34
Input chunk lengths: [128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128,
128, 128, 128, 128, 117, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 128, 41]
Chunk mapping: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1]
```

```python
def tokenize(element):
    outputs = tokenizer(
        element["content"],
        truncation=True,
        max_length=context_length,
        return_overflowing_tokens=True,
        return_length=True,
    )
    input_batch = []
    for length, input_ids in zip(outputs["length"], outputs["input_ids"]):
        if length == context_length:
            input_batch.append(input_ids)
    return {"input_ids": input_batch}


tokenized_datasets = raw_datasets.map(
    tokenize, batched=True, remove_columns=raw_datasets["train"].column_names
)
tokenized_datasets
```

```
DatasetDict({
    train: Dataset({
        features: ['input_ids'],
        num_rows: 16702061
    })
    valid: Dataset({
        features: ['input_ids'],
        num_rows: 93164
    })
})
```

```
[ ]  from transformers import AutoTokenizer, GPT2LMHeadModel, AutoConfig

     config = AutoConfig.from_pretrained(
         "gpt2",
         vocab_size=len(tokenizer),
         n_ctx=context_length,
         bos_token_id=tokenizer.bos_token_id,
         eos_token_id=tokenizer.eos_token_id,
     )
```

```
[ ]  model = GPT2LMHeadModel(config)
     model_size = sum(t.numel() for t in model.parameters())
     print(f"GPT-2 size: {model_size/1000**2:.1f}M parameters")
```

```
GPT-2 size: 124.2M parameters
```

```
[ ]  from transformers import DataCollatorForLanguageModeling

     tokenizer.pad_token = tokenizer.eos_token
     data_collator = DataCollatorForLanguageModeling(tokenizer, mlm=False)
```

```
[ ]  out = data_collator([tokenized_datasets["train"][i] for i in range(5)])
     for key in out:
         print(f"{key} shape: {out[key].shape}")
```

```
input_ids shape: torch.Size([5, 128])
attention_mask shape: torch.Size([5, 128])
labels shape: torch.Size([5, 128])
```

```
[ ]  from huggingface_hub import notebook_login

     notebook_login()
```

```
[ ]  from transformers import Trainer, TrainingArguments

     args = TrainingArguments(
         output_dir="codeparrot-ds",
         per_device_train_batch_size=32,
         per_device_eval_batch_size=32,
         evaluation_strategy="steps",
         eval_steps=5_000,
         logging_steps=5_000,
         gradient_accumulation_steps=8,
         num_train_epochs=1,
         weight_decay=0.1,
         warmup_steps=1_000,
         lr_scheduler_type="cosine",
         learning_rate=5e-4,
         save_steps=5_000,
         fp16=True,
         push_to_hub=True,
     )

     trainer = Trainer(
         model=model,
         tokenizer=tokenizer,
         args=args,
         data_collator=data_collator,
         train_dataset=tokenized_datasets["train"],
         eval_dataset=tokenized_datasets["valid"],
     )
```

```
[ ]  trainer.train()
```

```
[ ]  trainer.push_to_hub()
```

```
[ ] import torch
    from transformers import pipeline

    device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
    pipe = pipeline(
        "text-generation", model="huggingface-course/codeparrot-ds", device=device
    )
```

```
[ ] txt = """\
    # create some data
    x = np.random.randn(100)
    y = np.random.randn(100)

    # create scatter plot with x, y
    """
    print(pipe(txt, num_return_sequences=1)[0]["generated_text"])
```

```
    # create some data
    x = np.random.randn(100)
    y = np.random.randn(100)

    # create scatter plot with x, y
    plt.scatter(x, y)

    # create scatter
```

```
[ ] txt = """\
    # create some data
    x = np.random.randn(100)
    y = np.random.randn(100)

    # create dataframe from x and y
    """
    print(pipe(txt, num_return_sequences=1)[0]["generated_text"])
```

```
# create some data
x = np.random.randn(100)
y = np.random.randn(100)

# create dataframe from x and y
df = pd.DataFrame({'x': x, 'y': y})
df.insert(0,'x', x)
for
```

```
[ ] txt = """\
    # dataframe with profession, income and name
    df = pd.DataFrame({'profession': x, 'income':y, 'name': z})

    # calculate the mean income per profession
    """
    print(pipe(txt, num_return_sequences=1)[0]["generated_text"])
```

```
# dataframe with profession, income and name
df = pd.DataFrame({'profession': x, 'income':y, 'name': z})

# calculate the mean income per profession
profession = df.groupby(['profession']).mean()

# compute the
```

```
[ ] txt = """
    # import random forest regressor from scikit-learn
    from sklearn.ensemble import RandomForestRegressor

    # fit random forest model with 300 estimators on X, y:
    """
    print(pipe(txt, num_return_sequences=1)[0]["generated_text"])
```

```
    # import random forest regressor from scikit-learn
    from sklearn.ensemble import RandomForestRegressor

    # fit random forest model with 300 estimators on X, y:
    rf = RandomForestRegressor(n_estimators=300, random_state=random_state, max_depth=3)
    rf.fit(X, y)
    rf
```

```
[ ] keytoken_ids = []
    for keyword in [
        "plt",
        "pd",
        "sk",
        "fit",
        "predict",
        " plt",
        " pd",
        " sk",
        " fit",
        " predict",
        "testtest",
    ]:
        ids = tokenizer([keyword]).input_ids[0]
        if len(ids) == 1:
            keytoken_ids.append(ids[0])
        else:
            print(f"Keyword has not single token: {keyword}")
```

```
'Keyword has not single token: testtest'
```

```
[ ] from torch.nn import CrossEntropyLoss
    import torch


    def keytoken_weighted_loss(inputs, logits, keytoken_ids, alpha=1.0):
        # Shift so that tokens < n predict n
        shift_labels = inputs[..., 1:].contiguous()
        shift_logits = logits[..., :-1, :].contiguous()
        # Calculate per-token loss
        loss_fct = CrossEntropyLoss(reduce=False)
        loss = loss_fct(shift_logits.view(-1, shift_logits.size(-1)), shift_labels.view(-1))
        # Resize and average loss per sample
        loss_per_sample = loss.view(shift_logits.size(0), shift_logits.size(1)).mean(axis=1)
        # Calculate and scale weighting
        weights = torch.stack([(inputs == kt).float() for kt in keytoken_ids]).sum(
            axis=[0, 2]
        )
        weights = alpha * (1.0 + weights)
        # Calculate weighted average
        weighted_loss = (loss_per_sample * weights).mean()
        return weighted_loss
```

```
[ ] from torch.utils.data.dataloader import DataLoader

    tokenized_dataset.set_format("torch")
    train_dataloader = DataLoader(tokenized_dataset["train"], batch_size=32, shuffle=True)
    eval_dataloader = DataLoader(tokenized_dataset["valid"], batch_size=32)
```

```
[ ] weight_decay = 0.1


    def get_grouped_params(model, no_decay=["bias", "LayerNorm.weight"]):
```

```
[ ]     params_with_wd, params_without_wd = [], []
        for n, p in model.named_parameters():
            if any(nd in n for nd in no_decay):
                params_without_wd.append(p)
            else:
                params_with_wd.append(p)
        return [
            {"params": params_with_wd, "weight_decay": weight_decay},
            {"params": params_without_wd, "weight_decay": 0.0},
        ]
```

```
[ ] def evaluate():
        model.eval()
        losses = []
        for step, batch in enumerate(eval_dataloader):
            with torch.no_grad():
                outputs = model(batch["input_ids"], labels=batch["input_ids"])

            losses.append(accelerator.gather(outputs.loss))
        loss = torch.mean(torch.cat(losses))
        try:
            perplexity = torch.exp(loss)
        except OverflowError:
            perplexity = float("inf")
        return loss.item(), perplexity.item()
```

```
[ ] model = GPT2LMHeadModel(config)
```

```
[ ] from torch.optim import AdamW

    optimizer = AdamW(get_grouped_params(model), lr=5e-4)
```

```
[ ] from accelerate import Accelerator
```

```
[ ] accelerator = Accelerator(fp16=True)

    model, optimizer, train_dataloader, eval_dataloader = accelerator.prepare(
        model, optimizer, train_dataloader, eval_dataloader
    )
```

```
[ ] from transformers import get_scheduler

    num_train_epochs = 1
    num_update_steps_per_epoch = len(train_dataloader)
    num_training_steps = num_train_epochs * num_update_steps_per_epoch

    lr_scheduler = get_scheduler(
        name="linear",
        optimizer=optimizer,
        num_warmup_steps=1_000,
        num_training_steps=num_training_steps,
    )
```

```
[ ] from huggingface_hub import Repository, get_full_repo_name

    model_name = "codeparrot-ds-accelerate"
    repo_name = get_full_repo_name(model_name)
    repo_name
```

```
'sgugger/codeparrot-ds-accelerate'
```

```
[ ] output_dir = "codeparrot-ds-accelerate"
    repo = Repository(output_dir, clone_from=repo_name)
```

```
[ ] evaluate()
```

```
(10.934126853942871, 56057.14453125)
```

```
[ ]  from tqdm.notebook import tqdm

     gradient_accumulation_steps = 8
     eval_steps = 5_000

     model.train()
     completed_steps = 0
     for epoch in range(num_train_epochs):
         for step, batch in tqdm(
             enumerate(train_dataloader, start=1), total=num_training_steps
         ):
             logits = model(batch["input_ids"]).logits
             loss = keytoken_weighted_loss(batch["input_ids"], logits, keytoken_ids)
             if step % 100 == 0:
                 accelerator.print(
                     {
                         "lr": get_lr(),
                         "samples": step * samples_per_step,
                         "steps": completed_steps,
                         "loss/train": loss.item() * gradient_accumulation_steps,
                     }
                 )
             loss = loss / gradient_accumulation_steps
             accelerator.backward(loss)
             if step % gradient_accumulation_steps == 0:
                 accelerator.clip_grad_norm_(model.parameters(), 1.0)
                 optimizer.step()
                 lr_scheduler.step()
                 optimizer.zero_grad()
                 completed_steps += 1
             if (step % (eval_steps * gradient_accumulation_steps)) == 0:
                 eval_loss, perplexity = evaluate()
                 accelerator.print({"loss/eval": eval_loss, "perplexity": perplexity})
                 model.train()
                 accelerator.wait_for_everyone()
                 unwrapped_model = accelerator.unwrap_model(model)
                 unwrapped_model.save_pretrained(output_dir, save_function=accelerator.save)
                 if accelerator.is_main_process:
                     tokenizer.save_pretrained(output_dir)
                     repo.push_to_hub(
                         commit_message=f"Training in progress step {step}", blocking=False
                     )
```

8. How to Ask For Help

- What to do when you get an error

```
[ ] !pip install datasets evaluate transformers[sentencepiece]
    !apt install git-lfs
```

Kode ini menginstal library penting untuk proyek NLP, termasuk datasets untuk mengelola dataset, evaluate untuk mengevaluasi performa model, dan transformers[sentencepiece] untuk bekerja dengan model-model canggih dari Hugging Face, khususnya yang menggunakan SentencePiece tokenizer. Selain itu, kode ini juga menginstal Git LFS untuk menangani file besar, seperti model terlatih dan dataset, yang digunakan dalam repositori Git, terutama saat bekerja dengan Hugging Face Hub.

```
[ ] !git config --global user.email "m.rizkyf400@gmail.com"
    !git config --global user.name "Rizkyy3"
```

Kode ini digunakan untuk mengatur konfigurasi global Git dengan menetapkan email dan nama pengguna. Konfigurasi ini memastikan setiap kontribusi pada repositori Git memiliki identitas yang jelas, memudahkan pelacakan perubahan, dan menjaga keterhubungan dengan platform pengelolaan versi seperti GitHub atau GitLab.

```
[ ] from huggingface_hub import notebook_login

    notebook_login()
```

Kode ini digunakan untuk autentikasi ke Hugging Face Hub melalui notebook. Dengan menjalankan notebook_login(), pengguna dapat memasukkan token akses mereka untuk mendapatkan izin mengelola model, dataset, dan sumber daya lain di Hugging Face Hub secara langsung dari lingkungan notebook, sehingga mempermudah integrasi dan kolaborasi dalam proyek NLP.

```
[ ] from distutils.dir_util import copy_tree
    from huggingface_hub import Repository, snapshot_download, create_repo, get_full_repo_name


    def copy_repository_template():
        # Clone the repo and extract the local path
        template_repo_id = "lewtun/distilbert-base-uncased-finetuned-squad-d5716d28"
        commit_hash = "be3eaffc28669d7932492681cd5f3e8905e358b4"
        template_repo_dir = snapshot_download(template_repo_id, revision=commit_hash)
        # Create an empty repo on the Hub
        model_name = template_repo_id.split("/")[1]
        create_repo(model_name, exist_ok=True)
        # Clone the empty repo
        new_repo_id = get_full_repo_name(model_name)
        new_repo_dir = model_name
        repo = Repository(local_dir=new_repo_dir, clone_from=new_repo_id)
        # Copy files
        copy_tree(template_repo_dir, new_repo_dir)
        # Push to Hub
        repo.push_to_hub()
```

```
[ ] from transformers import pipeline

    model_checkpoint = get_full_repo_name("distillbert-base-uncased-finetuned-squad-d5716d28")
    reader = pipeline("question-answering", model=model_checkpoint)
```

```
"""
OSError: Can't load config for 'lewtun/distillbert-base-uncased-finetuned-squad-d5716d28'. Make
sure that:

- 'lewtun/distillbert-base-uncased-finetuned-squad-d5716d28' is a correct model identifier listed
on 'https://huggingface.co/models'

- or 'lewtun/distillbert-base-uncased-finetuned-squad-d5716d28' is the correct path to a directory
containing a config.json file
"""
```

```
[ ] model_checkpoint = get_full_repo_name("distilbert-base-uncased-finetuned-squad-d5716d28")
    reader = pipeline("question-answering", model=model_checkpoint)
```

```
"""
OSError: Can't load config for 'lewtun/distilbert-base-uncased-finetuned-squad-d5716d28'. Make
sure that:

- 'lewtun/distilbert-base-uncased-finetuned-squad-d5716d28' is a correct model identifier listed
on 'https://huggingface.co/models'

- or 'lewtun/distilbert-base-uncased-finetuned-squad-d5716d28' is the correct path to a directory
containing a config.json file
"""
```

```
[ ] from huggingface_hub import list_repo_files

    list_repo_files(repo_id=model_checkpoint)
```

```
['.gitattributes', 'README.md', 'pytorch_model.bin', 'special_tokens_map.json',
'tokenizer_config.json', 'training_args.bin', 'vocab.txt']
```

```
[ ] from transformers import AutoConfig

    pretrained_checkpoint = "distilbert-base-uncased"
    config = AutoConfig.from_pretrained(pretrained_checkpoint)
```

```
[ ] config.push_to_hub(model_checkpoint, commit_message="Add config.json")
```

```
[ ] reader = pipeline("question-answering", model=model_checkpoint, revision="main")

    context = r"""
    Extractive Question Answering is the task of extracting an answer from a text
    given a question. An example of a question answering dataset is the SQuAD
    dataset, which is entirely based on that task. If you would like to fine-tune a
    model on a SQuAD task, you may leverage the
    examples/pytorch/question-answering/run_squad.py script.

    🤗 Transformers is interoperable with the PyTorch, TensorFlow, and JAX
    frameworks, so you can use your favourite tools for a wide variety of tasks!
    """

    question = "What is extractive question answering?"
    reader(question=question, context=context)
```

```
{'score': 0.38669535517692566,
 'start': 34,
 'end': 95,
 'answer': 'the task of extracting an answer from a text given a question'}
```

```
[ ] tokenizer = reader.tokenizer
    model = reader.model
```

```
[ ] question = "Which frameworks can I use?"
```

```
[ ]
```

```
attention_mask, head_mask, inputs_embeds, start_positions, end_positions, output_attentions,
output_hidden_states, return_dict)
    723         return_dict = return_dict if return_dict is not None else
self.config.use_return_dict
    724
--> 725         distilbert_output = self.distilbert(
    726             input_ids=input_ids,
    727             attention_mask=attention_mask,

~/miniconda3/envs/huggingface/lib/python3.8/site-packages/torch/nn/modules/module.py in
_call_impl(self, *input, **kwargs)
   1049         if not (self._backward_hooks or self._forward_hooks or self._forward_pre_hooks or
_global_backward_hooks
   1050                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1051             return forward_call(*input, **kwargs)
   1052         # Do not call functions when jit is used
   1053         full_backward_hooks, non_full_backward_hooks = [], []

~/miniconda3/envs/huggingface/lib/python3.8/site-
packages/transformers/models/distilbert/modeling_distilbert.py in forward(self, input_ids,
attention_mask, head_mask, inputs_embeds, output_attentions, output_hidden_states, return_dict)
    471             raise ValueError("You cannot specify both input_ids and inputs_embeds at the
same time")
    472         elif input_ids is not None:
--> 473             input_shape = input_ids.size()
    474         elif inputs_embeds is not None:
    475             input_shape = inputs_embeds.size()[:-1]

AttributeError: 'list' object has no attribute 'size'
"""
```

```
[ ]  inputs["input_ids"][:5]
```

```
[101, 2029, 7705, 2015, 2064]
```

```
[ ]  type(inputs["input_ids"])
```

```
list
```

```
[ ]  import torch

     inputs = tokenizer(question, context, add_special_tokens=True)
     input_ids = inputs["input_ids"][0]
     outputs = model(**inputs)
     answer_start_scores = outputs.start_logits
     answer_end_scores = outputs.end_logits
     # Get the most likely beginning of answer with the argmax of the score
     answer_start = torch.argmax(answer_start_scores)
     # Get the most likely end of answer with the argmax of the score
     answer_end = torch.argmax(answer_end_scores) + 1
     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}")
```

```
     """
     ---------------------------------------------------------------------------
     AttributeError                            Traceback (most recent call last)
     /var/folders/28/k4cy5q7s2hs92xq7_h89_vgm0000gn/T/ipykernel_75743/2725838073.py in <modul
           1 inputs = tokenizer(question, text, add_special_tokens=True)
           2 input_ids = inputs["input_ids"]
     ----> 3 outputs = model(**inputs)
           4 answer_start_scores = outputs.start_logits
           5 answer_end_scores = outputs.end_logits

     ~/miniconda3/envs/huggingface/lib/python3.8/site-packages/torch/nn/modules/module.py in
     _call_impl(self, *input, **kwargs)
        1049         if not (self._backward_hooks or self._forward_hooks or self._forward_pre
     _global_backward_hooks
        1050                 or _global_forward_hooks or _global_forward_pre_hooks):
     -> 1051             return forward_call(*input, **kwargs)
        1052         # Do not call functions when jit is used
        1053         full_backward_hooks, non_full_backward_hooks = [], []

     ~/miniconda3/envs/huggingface/lib/python3.8/site-
     packages/transformers/models/distilbert/modeling_distilbert.py in forward(self, input_ic
```

```
[ ]
```

```
attention_mask, head_mask, inputs_embeds, start_positions, end_positions, output_attentions,
output_hidden_states, return_dict)
    723         return_dict = return_dict if return_dict is not None else
self.config.use_return_dict
    724
--> 725         distilbert_output = self.distilbert(
    726             input_ids=input_ids,
    727             attention_mask=attention_mask,

~/miniconda3/envs/huggingface/lib/python3.8/site-packages/torch/nn/modules/module.py in
_call_impl(self, *input, **kwargs)
   1049         if not (self._backward_hooks or self._forward_hooks or self._forward_pre_hooks or
_global_backward_hooks
   1050                 or _global_forward_hooks or _global_forward_pre_hooks):
-> 1051             return forward_call(*input, **kwargs)
   1052         # Do not call functions when jit is used
   1053         full_backward_hooks, non_full_backward_hooks = [], []

~/miniconda3/envs/huggingface/lib/python3.8/site-
packages/transformers/models/distilbert/modeling_distilbert.py in forward(self, input_ids,
attention_mask, head_mask, inputs_embeds, output_attentions, output_hidden_states, return_dict)
    471             raise ValueError("You cannot specify both input_ids and inputs_embeds at the
same time")
    472         elif input_ids is not None:
--> 473             input_shape = input_ids.size()
    474         elif inputs_embeds is not None:
    475             input_shape = inputs_embeds.size()[:-1]

AttributeError: 'list' object has no attribute 'size'
"""
```

```
[ ]  inputs["input_ids"][:5]
```

```
[101, 2029, 7705, 2015, 2064]
```

```
[ ]  type(inputs["input_ids"])
```

```
list
```

- Asking for help on the forums

```
[ ] !pip install datasets evaluate transformers[sentencepiece]
```

Kode ini menginstal library utama untuk proyek NLP, yaitu datasets untuk memuat dan memproses dataset, evaluate untuk mengukur performa model, dan transformers[sentencepiece] untuk bekerja dengan model-model canggih dari Hugging Face yang menggunakan SentencePiece tokenizer, memungkinkan pengguna untuk melakukan berbagai tugas NLP secara efisien.

```
[ ] from transformers import AutoTokenizer, AutoModel

    model_checkpoint = "distilbert-base-uncased"
    tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
    model = AutoModel.from_pretrained(model_checkpoint)
```

```
[ ] text = """
    Generation One is a retroactive term for the Transformers characters that
    appeared between 1984 and 1993. The Transformers began with the 1980s Japanese
    toy lines Micro Change and Diaclone. They presented robots able to transform
    into everyday vehicles, electronic items or weapons. Hasbro bought the Micro
    Change and Diaclone toys, and partnered with Takara. Marvel Comics was hired by
    Hasbro to create the backstory; editor-in-chief Jim Shooter wrote an overall
    story, and gave the task of creating the characthers to writer Dennis O'Neil.
    Unhappy with O'Neil's work (although O'Neil created the name "Optimus Prime"),
    Shooter chose Bob Budiansky to create the characters.

    The Transformers mecha were largely designed by Shōji Kawamori, the creator of
    the Japanese mecha anime franchise Macross (which was adapted into the Robotech
    franchise in North America). Kawamori came up with the idea of transforming
    mechs while working on the Diaclone and Macross franchises in the early 1980s
    (such as the VF-1 Valkyrie in Macross and Robotech), with his Diaclone mechs
    later providing the basis for Transformers.

    The primary concept of Generation One is that the heroic Optimus Prime, the
```

```
[ ]  villainous Megatron, and their finest soldiers crash land on pre-historic Earth
     in the Ark and the Nemesis before awakening in 1985, Cybertron hurtling through
     the Neutral zone as an effect of the war. The Marvel comic was originally part
     of the main Marvel Universe, with appearances from Spider-Man and Nick Fury,
     plus some cameos, as well as a visit to the Savage Land.

     The Transformers TV series began around the same time. Produced by Sunbow
     Productions and Marvel Productions, later Hasbro Productions, from the start it
     contradicted Budiansky's backstories. The TV series shows the Autobots looking
     for new energy sources, and crash landing as the Decepticons attack. Marvel
     interpreted the Autobots as destroying a rogue asteroid approaching Cybertron.
     Shockwave is loyal to Megatron in the TV series, keeping Cybertron in a
     stalemate during his absence, but in the comic book he attempts to take command
     of the Decepticons. The TV series would also differ wildly from the origins
     Budiansky had created for the Dinobots, the Decepticon turned Autobot Jetfire
     (known as Skyfire on TV), the Constructicons (who combine to form
     Devastator),[19][20] and Omega Supreme. The Marvel comic establishes early on
     that Prime wields the Creation Matrix, which gives life to machines. In the
     second season, the two-part episode The Key to Vector Sigma introduced the
     ancient Vector Sigma computer, which served the same original purpose as the
     Creation Matrix (giving life to Transformers), and its guardian Alpha Trion.
     """

     inputs = tokenizer(text, return_tensors="pt")
     logits = model(**inputs).logits
```

- Debugging the training pipeline

```
[ ] !pip install datasets evaluate transformers[sentencepiece]
```

Kode ini menginstal library utama untuk proyek NLP, yaitu datasets untuk memuat dan memproses dataset, evaluate untuk mengukur performa model, dan transformers[sentencepiece] untuk bekerja dengan model-model canggih dari Hugging Face yang menggunakan SentencePiece tokenizer, memungkinkan pengguna untuk melakukan berbagai tugas NLP secara efisien.

```
[ ] from datasets import load_dataset
    import evaluate
    from transformers import (
        AutoTokenizer,
        AutoModelForSequenceClassification,
        TrainingArguments,
        Trainer,
    )

    raw_datasets = load_dataset("glue", "mnli")

    model_checkpoint = "distilbert-base-uncased"
    tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)


    def preprocess_function(examples):
        return tokenizer(examples["premise"], examples["hypothesis"], truncation=True)


    tokenized_datasets = raw_datasets.map(preprocess_function, batched=True)
    model = AutoModelForSequenceClassification.from_pretrained(model_checkpoint)

    args = TrainingArguments(
        f"distilbert-finetuned-mnli",
        evaluation_strategy="epoch",
        save_strategy="epoch",
        learning_rate 2e-5
```

```
[ ]     learning_rate=2e-5,
        num_train_epochs=3,
        weight_decay=0.01,
    )

    metric = evaluate.load("glue", "mnli")


    def compute_metrics(eval_pred):
        predictions, labels = eval_pred
        return metric.compute(predictions=predictions, references=labels)


    trainer = Trainer(
        model,
        args,
        train_dataset=raw_datasets["train"],
        eval_dataset=raw_datasets["validation_matched"],
        compute_metrics=compute_metrics,
    )
    trainer.train()
```

```
'ValueError: You have to specify either input_ids or inputs_embeds'
```

```
[ ] trainer.train_dataset[0]
```

```
{'hypothesis': 'Product and geography are what make cream skimming work. ',
 'idx': 0,
 'label': 1,
 'premise': 'Conceptually cream skimming has two basic dimensions - product and geography.'}
```

```
[ ]  from datasets import load_dataset
     import evaluate
     from transformers import (
         AutoTokenizer,
         AutoModelForSequenceClassification,
         TrainingArguments,
         Trainer,
     )

     raw_datasets = load_dataset("glue", "mnli")

     model_checkpoint = "distilbert-base-uncased"
     tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)


     def preprocess_function(examples):
         return tokenizer(examples["premise"], examples["hypothesis"], truncation=True)


     tokenized_datasets = raw_datasets.map(preprocess_function, batched=True)
     model = AutoModelForSequenceClassification.from_pretrained(model_checkpoint)

     args = TrainingArguments(
         f"distilbert-finetuned-mnli",
         evaluation_strategy="epoch",
         save_strategy="epoch",
         learning_rate=2e-5,
         num_train_epochs=3,
         weight_decay=0.01,
     )

     metric = evaluate.load("glue", "mnli")


     def compute_metrics(eval_pred):
         predictions, labels = eval_pred
```

```
[ ]     predictions, labels = eval_pred
        return metric.compute(predictions=predictions, references=labels)


    trainer = Trainer(
        model,
        args,
        train_dataset=tokenized_datasets["train"],
        eval_dataset=tokenized_datasets["validation_matched"],
        compute_metrics=compute_metrics,
    )
    trainer.train()
```

```
'ValueError: expected sequence of length 43 at dim 1 (got 37)'
```

```
[ ] tokenizer.decode(trainer.train_dataset[0]["input_ids"])
```

```
'[CLS] conceptually cream skimming has two basic dimensions - product and geography. [SEP] product
and geography are what make cream skimming work. [SEP]'
```

```
[ ] trainer.train_dataset[0].keys()
```

```
dict_keys(['attention_mask', 'hypothesis', 'idx', 'input_ids', 'label', 'premise'])
```

```
[ ] type(trainer.model)
```

```
transformers.models.distilbert.modeling_distilbert.DistilBertForSequenceClassification
```

```
[ ] trainer.train_dataset[0]["attention_mask"]
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```
[ ] len(trainer.train_dataset[0]["attention_mask"]) == len(
        trainer.train_dataset[0]["input_ids"]
    )
```

```
True
```

```
[ ] trainer.train_dataset[0]["label"]
```

```
1
```

```
[ ] trainer.train_dataset.features["label"].names
```

```
['entailment', 'neutral', 'contradiction']
```

```
[ ] for batch in trainer.get_train_dataloader():
        break
```

```
~/git/transformers/src/transformers/data/data_collator.py in torch_default_data_collator(features)
    105                 batch[k] = torch.stack([f[k] for f in features])
    106             else:
--> 107                 batch[k] = torch.tensor([f[k] for f in features])
    108
    109     return batch

ValueError: expected sequence of length 45 at dim 1 (got 76)
```

```
[ ] data_collator = trainer.get_train_dataloader().collate_fn
    data_collator
```

```
<function transformers.data.data_collator.default_data_collator(features: List[InputDataClass],
return_tensors='pt') -> Dict[str, Any]>
```

```
[ ]  from datasets import load_dataset
     import evaluate
     from transformers import (
         AutoTokenizer,
         AutoModelForSequenceClassification,
         DataCollatorWithPadding,
         TrainingArguments,
         Trainer,
     )

     raw_datasets = load_dataset("glue", "mnli")

     model_checkpoint = "distilbert-base-uncased"
     tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)


     def preprocess_function(examples):
         return tokenizer(examples["premise"], examples["hypothesis"], truncation=True)


     tokenized_datasets = raw_datasets.map(preprocess_function, batched=True)
     model = AutoModelForSequenceClassification.from_pretrained(model_checkpoint)

     args = TrainingArguments(
         f"distilbert-finetuned-mnli",
         evaluation_strategy="epoch",
         save_strategy="epoch",
         learning_rate=2e-5,
         num_train_epochs=3,
         weight_decay=0.01,
     )

     metric = evaluate.load("glue", "mnli")


     def compute_metrics(eval_pred):
```

```
[ ] def compute_metrics(eval_pred):
        predictions, labels = eval_pred
        return metric.compute(predictions=predictions, references=labels)


    data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

    trainer = Trainer(
        model,
        args,
        train_dataset=tokenized_datasets["train"],
        eval_dataset=tokenized_datasets["validation_matched"],
        compute_metrics=compute_metrics,
        data_collator=data_collator,
        tokenizer=tokenizer,
    )
    trainer.train()
```

```
RuntimeError: CUDA error: CUBLAS_STATUS_ALLOC_FAILED when calling `cublasCreate(handle)`
```

```
[ ] data_collator = trainer.get_train_dataloader().collate_fn
    batch = data_collator([trainer.train_dataset[i] for i in range(4)])
```

```
[ ] data_collator = trainer.get_train_dataloader().collate_fn
    actual_train_set = trainer._remove_unused_columns(trainer.train_dataset)
    batch = data_collator([actual_train_set[i] for i in range(4)])
```

```
[ ] for batch in trainer.get_train_dataloader():
        break
```

```
[ ] outputs = trainer.model.cpu()(**batch)
```

```
~/.pyenv/versions/3.7.9/envs/base/lib/python3.7/site-packages/torch/nn/functional.py in
nll_loss(input, target, weight, size_average, ignore_index, reduce, reduction)
   2386         )
   2387     if dim == 2:
-> 2388         ret = torch._C._nn.nll_loss(input, target, weight, _Reduction.get_enum(reduction),
ignore_index)
   2389     elif dim == 4:
   2390         ret = torch._C._nn.nll_loss2d(input, target, weight,
_Reduction.get_enum(reduction), ignore_index)

IndexError: Target 2 is out of bounds.
```

```
[ ] trainer.model.config.num_labels
```

```
2
```

```
[ ] from datasets import load_dataset
    import evaluate
    from transformers import (
        AutoTokenizer,
        AutoModelForSequenceClassification,
        DataCollatorWithPadding,
        TrainingArguments,
        Trainer,
    )

    raw_datasets = load_dataset("glue", "mnli")

    model_checkpoint = "distilbert-base-uncased"
    tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)


    def preprocess_function(examples):
```

```
[ ]     return tokenizer(examples["premise"], examples["hypothesis"], truncation=True)


    tokenized_datasets = raw_datasets.map(preprocess_function, batched=True)
    model = AutoModelForSequenceClassification.from_pretrained(model_checkpoint, num_labels=3)

    args = TrainingArguments(
        f"distilbert-finetuned-mnli",
        evaluation_strategy="epoch",
        save_strategy="epoch",
        learning_rate=2e-5,
        num_train_epochs=3,
        weight_decay=0.01,
    )

    metric = evaluate.load("glue", "mnli")


    def compute_metrics(eval_pred):
        predictions, labels = eval_pred
        return metric.compute(predictions=predictions, references=labels)


    data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

    trainer = Trainer(
        model,
        args,
        train_dataset=tokenized_datasets["train"],
        eval_dataset=tokenized_datasets["validation_matched"],
        compute_metrics=compute_metrics,
        data_collator=data_collator,
        tokenizer=tokenizer,
    )
```

```
[ ] for batch in trainer.get_train_dataloader():
        break

    outputs = trainer.model.cpu()(**batch)
```

```
    import torch

    device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
    batch = {k: v.to(device) for k, v in batch.items()}

    outputs = trainer.model.to(device)(**batch)
```

```
[ ] loss = outputs.loss
    loss.backward()
```

```
[ ] trainer.create_optimizer()
    trainer.optimizer.step()
```

```
[ ] # This will take a long time and error out, so you shouldn't run this cell
    trainer.train()
```

```
TypeError: only size-1 arrays can be converted to Python scalars
```

```
[ ] trainer.evaluate()
```

```
TypeError: only size-1 arrays can be converted to Python scalars
```

```
[ ] for batch in trainer.get_eval_dataloader():
        break

    batch = {k: v.to(device) for k, v in batch.items()}
```

```
[ ] with torch.no_grad():
        outputs = trainer.model(**batch)
```

```
[ ] predictions = outputs.logits.cpu().numpy()
    labels = batch["labels"].cpu().numpy()

    compute_metrics((predictions, labels))
```

```
TypeError: only size-1 arrays can be converted to Python scalars
```

```
[ ] predictions.shape, labels.shape
```

```
((8, 3), (8,))
```

```
[ ] import numpy as np


    def compute_metrics(eval_pred):
        predictions, labels = eval_pred
        predictions = np.argmax(predictions, axis=1)
        return metric.compute(predictions=predictions, references=labels)


    compute_metrics((predictions, labels))
```

```
{'accuracy': 0.625}
```

```
[ ]  import numpy as np
     from datasets import load_dataset
     import evaluate
     from transformers import (
         AutoTokenizer,
         AutoModelForSequenceClassification,
         DataCollatorWithPadding,
         TrainingArguments,
         Trainer,
     )

     raw_datasets = load_dataset("glue", "mnli")

     model_checkpoint = "distilbert-base-uncased"
     tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)


     def preprocess_function(examples):
         return tokenizer(examples["premise"], examples["hypothesis"], truncation=True)


     tokenized_datasets = raw_datasets.map(preprocess_function, batched=True)
     model = AutoModelForSequenceClassification.from_pretrained(model_checkpoint, num_labels=3)

     args = TrainingArguments(
         f"distilbert-finetuned-mnli",
         evaluation_strategy="epoch",
         save_strategy="epoch",
         learning_rate=2e-5,
         num_train_epochs=3,
         weight_decay=0.01,
     )

     metric = evaluate.load("glue", "mnli")
```

```
[ ] def compute_metrics(eval_pred):
        predictions, labels = eval_pred
        predictions = np.argmax(predictions, axis=1)
        return metric.compute(predictions=predictions, references=labels)


    data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

    trainer = Trainer(
        model,
        args,
        train_dataset=tokenized_datasets["train"],
        eval_dataset=tokenized_datasets["validation_matched"],
        compute_metrics=compute_metrics,
        data_collator=data_collator,
        tokenizer=tokenizer,
    )
    trainer.train()
```

```
[ ] for batch in trainer.get_train_dataloader():
        break

    batch = {k: v.to(device) for k, v in batch.items()}
    trainer.create_optimizer()

    for _ in range(20):
        outputs = trainer.model(**batch)
        loss = outputs.loss
        loss.backward()
        trainer.optimizer.step()
        trainer.optimizer.zero_grad()
```

```
[ ] with torch.no_grad():
        outputs = trainer.model(**batch)
    preds = outputs.logits
    labels = batch["labels"]

    compute_metrics((preds.cpu().numpy(), labels.cpu().numpy()))
```

```
{'accuracy': 1.0}
```

9. Building and Sharing Demos

- **Building your first demo**

```
[ ] !pip install gradio
```

```
[ ] import gradio as gr


    def greet(name):
        return "Hello " + name


    demo = gr.Interface(fn=greet, inputs="text", outputs="text")

    demo.launch()
```

```
[ ] import gradio as gr


    def greet(name):
        return "Hello " + name


    # We instantiate the Textbox class
    textbox = gr.Textbox(label="Type your name here:", placeholder="John Doe", lines=2)

    gr.Interface(fn=greet, inputs=textbox, outputs="text").launch()
```

```
[ ] from transformers import pipeline

    model = pipeline("text-generation")


    def predict(prompt):
        completion = model(prompt)[0]["generated_text"]
        return completion
```

```
[ ] import gradio as gr

    gr.Interface(fn=predict, inputs="text", outputs="text").launch()
```

- Understanding the Interface class

```
[ ] !pip install datasets evaluate transformers[sentencepiece]
```

Kode ini menginstal library yang diperlukan untuk pengembangan proyek NLP, termasuk datasets untuk memuat dan mengelola dataset, evaluate untuk mengevaluasi kinerja model, dan transformers[sentencepiece] untuk bekerja dengan model-model canggih dari Hugging Face yang menggunakan SentencePiece tokenizer, sehingga memudahkan implementasi dan eksperimen dengan berbagai model bahasa alami.

```
[ ] !pip install gradio
```

Kode ini menginstal Gradio, sebuah library Python yang memungkinkan pengguna untuk dengan mudah membuat antarmuka web interaktif untuk model machine learning, termasuk model NLP, sehingga memudahkan dalam demo, eksperimen, dan berbagi model dengan orang lain secara cepat dan intuitif.

```
[ ] import numpy as np
    import gradio as gr


    def reverse_audio(audio):
        sr, data = audio
        reversed_audio = (sr, np.flipud(data))
        return reversed_audio


    mic = gr.Audio(source="microphone", type="numpy", label="Speak here...")
    gr.Interface(reverse_audio, mic, "audio").launch()
```

```
[ ]  import numpy as np
     import gradio as gr

     notes = ["C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B"]


     def generate_tone(note, octave, duration):
         sr = 48000
         a4_freq, tones_from_a4 = 440, 12 * (octave - 4) + (note - 9)
         frequency = a4_freq * 2 ** (tones_from_a4 / 12)
         duration = int(duration)
         audio = np.linspace(0, duration, duration * sr)
         audio = (20000 * np.sin(audio * (2 * np.pi * frequency))).astype(np.int16)
         return (sr, audio)


     gr.Interface(
         generate_tone,
         [
             gr.Dropdown(notes, type="index"),
             gr.Slider(minimum=4, maximum=6, step=1),
             gr.Textbox(type="number", value=1, label="Duration in seconds"),
         ],
         "audio",
     ).launch()
```

```
[ ]  from transformers import pipeline
     import gradio as gr

     model = pipeline("automatic-speech-recognition")


     def transcribe_audio(mic=None, file=None):
         if mic is not None:
             audio = mic
         elif file is not None:
             audio = file
         else:
             return "You must either provide a mic recording or a file"
         transcription = model(audio)["text"]
         return transcription


     gr.Interface(
         fn=transcribe_audio,
         inputs=[
             gr.Audio(source="microphone", type="filepath", optional=True),
             gr.Audio(source="upload", type="filepath", optional=True),
         ],
         outputs="text",
     ).launch()
```

- Sharing demos with others

```
[ ]  !pip install datasets evaluate transformers[sentencepiece]
```

Kode ini menginstal library yang diperlukan untuk pengembangan proyek NLP, termasuk datasets untuk memuat dan mengelola dataset, evaluate untuk mengevaluasi kinerja model, dan transformers[sentencepiece] untuk bekerja dengan model-model canggih dari Hugging Face yang menggunakan SentencePiece tokenizer, sehingga memudahkan implementasi dan eksperimen dengan berbagai model bahasa alami.

```
[ ]  !pip install gradio
```

Kode ini menginstal Gradio, sebuah library Python yang memungkinkan pengguna untuk dengan mudah membuat antarmuka web interaktif untuk model machine learning, termasuk model NLP, sehingga memudahkan dalam demo, eksperimen, dan berbagi model dengan orang lain secara cepat dan intuitif.

```
[ ]  title = "Ask Rick a Question"
     description = """
     The bot was trained to answer questions based on Rick and Morty dialogues. Ask Rick anything!
     <img src="https://huggingface.co/spaces/course-demos/Rick_and_Morty_QA/resolve/main/rick.png" wid
     """

     article = "Check out [the original Rick and Morty Bot](https://huggingface.co/spaces/kingabzpro/R

     gr.Interface(
         fn=predict,
         inputs="textbox",
         outputs="text",
         title=title,
         description=description,
         article=article,
         examples=[["What are you doing?"], ["Where should we time travel to?"]],
     ).launch()
```

```
[ ]  gr.Interface(classify_image, "image", "label").launch(share=True)
```

```
[ ]  from pathlib import Path
     import torch
     import gradio as gr
     from torch import nn

     LABELS = Path("class_names.txt").read_text().splitlines()

     model = nn.Sequential(
         nn.Conv2d(1, 32, 3, padding="same"),
         nn.ReLU(),
         nn.MaxPool2d(2),
         nn.Conv2d(32, 64, 3, padding="same"),
         nn.ReLU(),
         nn.MaxPool2d(2),
         nn.Conv2d(64, 128, 3, padding="same"),
         nn.ReLU(),
         nn.MaxPool2d(2),
         nn.Flatten(),
         nn.Linear(1152, 256),
         nn.ReLU(),
         nn.Linear(256, len(LABELS)),
     )
     state_dict = torch.load("pytorch_model.bin", map_location="cpu")
     model.load_state_dict(state_dict, strict=False)
     model.eval()


     def predict(im):
         x = torch.tensor(im, dtype=torch.float32).unsqueeze(0).unsqueeze(0) / 255.0
         with torch.no_grad():
             out = model(x)
         probabilities = torch.nn.functional.softmax(out[0], dim=0)
         values, indices = torch.topk(probabilities, 5)
```

```
[ ]      return {LABELS[i]: v.item() for i, v in zip(indices, values)}
```

```
[ ]  interface = gr.Interface(
         predict,
         inputs="sketchpad",
         outputs="label",
         theme="huggingface",
         title="Sketch Recognition",
         description="Who wants to play Pictionary? Draw a common object like a shovel or a laptop, and
         article="<p style='text-align: center'>Sketch Recognition | Demo Model</p>",
         live=True,
     )
     interface.launch(share=True)
```

- **Integrations with the Hugging Face Hub**

```
[ ]  !pip install datasets evaluate transformers[sentencepiece]
```

Kode ini menginstal library yang diperlukan untuk pengembangan proyek NLP, termasuk datasets untuk memuat dan mengelola dataset, evaluate untuk mengevaluasi kinerja model, dan transformers[sentencepiece] untuk bekerja dengan model-model canggih dari Hugging Face yang menggunakan SentencePiece tokenizer, sehingga memudahkan implementasi dan eksperimen dengan berbagai model bahasa alami.

```
[ ]  !pip install gradio
```

Kode ini menginstal Gradio, sebuah library Python yang memungkinkan pengguna untuk dengan mudah membuat antarmuka web interaktif untuk model machine learning, termasuk model NLP, sehingga memudahkan dalam demo, eksperimen, dan berbagi model dengan orang lain secara cepat dan intuitif.

```
[ ]  import gradio as gr

     title = "GPT-J-6B"
     description = "Gradio Demo for GPT-J 6B, a transformer model trained using Ben Wang's Mesh Transfo
     article = "<p style='text-align: center'><a href='https://github.com/kingoflolz/mesh-transformer-
     examples = [
         ["The tower is 324 metres (1,063 ft) tall,"],
         ["The Moon's orbit around Earth has"],
         ["The smooth Borealis basin in the Northern Hemisphere covers 40%"],
     ]
     gr.Interface.load(
         "huggingface/EleutherAI/gpt-j-6B",
         inputs=gr.Textbox(lines=5, label="Input Text"),
         title=title,
         description=description,
         article=article,
         examples=examples,
         enable_queue=True,
     ).launch()
```

```
[ ]  gr.Interface.load("spaces/abidlabs/remove-bg").launch()
```

```
[ ]  gr.Interface.load(
         "spaces/abidlabs/remove-bg", inputs="webcam", title="Remove your webcam background!"
     ).launch()
```

- **Advanced Interface features**

```
[ ]  !pip install datasets evaluate transformers[sentencepiece]
```

Kode ini menginstal library yang diperlukan untuk pengembangan proyek NLP, termasuk datasets untuk memuat dan mengelola dataset, evaluate untuk mengevaluasi kinerja model, dan transformers[sentencepiece] untuk bekerja dengan model-model canggih dari Hugging Face yang menggunakan SentencePiece tokenizer, sehingga memudahkan implementasi dan eksperimen dengan berbagai model bahasa alami.

```
[ ]  !pip install gradio
```

Kode ini menginstal Gradio, sebuah library Python yang memungkinkan pengguna untuk dengan mudah membuat antarmuka web interaktif untuk model machine learning, termasuk model NLP, sehingga memudahkan dalam demo, eksperimen, dan berbagi model dengan orang lain secara cepat dan intuitif.

```
[ ]  import random

     import gradio as gr


     def chat(message, history):
         history = history or []
         if message.startswith("How many"):
             response = random.randint(1, 10)
         elif message.startswith("How"):
             response = random.choice(["Great", "Good", "Okay", "Bad"])
         elif message.startswith("Where"):
             response = random.choice(["Here", "There", "Somewhere"])
         else:
             response = "I don't know"
         history.append((message, response))
         return history, history
```

```
[ ]  iface = gr.Interface(
         chat,
         ["text", "state"],
         ["chatbot", "state"],
         allow_screenshot=False,
         allow_flagging="never",
     )
     iface.launch()
```

```
[ ]  import requests
     import tensorflow as tf

     import gradio as gr

     inception_net = tf.keras.applications.MobileNetV2()  # load the model

     # Download human-readable labels for ImageNet.
     response = requests.get("https://git.io/JJkYN")
     labels = response.text.split("\n")


     def classify_image(inp):
         inp = inp.reshape((-1, 224, 224, 3))
         inp = tf.keras.applications.mobilenet_v2.preprocess_input(inp)
         prediction = inception_net.predict(inp).flatten()
         return {labels[i]: float(prediction[i]) for i in range(1000)}


     image = gr.Image(shape=(224, 224))
     label = gr.Label(num_top_classes=3)

     title = "Gradio Image Classifiction + Interpretation Example"
     gr.Interface(
         fn=classify_image, inputs=image, outputs=label, interpretation="default", title=title
     ).launch()
```

- **Introduction to Blocks**

```
[ ]  !pip install datasets evaluate transformers[sentencepiece]
```

Kode ini menginstal library yang diperlukan untuk pengembangan proyek NLP, termasuk datasets untuk memuat dan mengelola dataset, evaluate untuk mengevaluasi kinerja model, dan transformers[sentencepiece] untuk bekerja dengan model-model canggih dari Hugging Face yang menggunakan SentencePiece tokenizer, sehingga memudahkan implementasi dan eksperimen dengan berbagai model bahasa alami.

```
[ ]  !pip install gradio
```

Kode ini menginstal Gradio, sebuah library Python yang memungkinkan pengguna untuk dengan mudah membuat antarmuka web interaktif untuk model machine learning, termasuk model NLP, sehingga memudahkan dalam demo, eksperimen, dan berbagi model dengan orang lain secara cepat dan intuitif.

```
[ ]  import gradio as gr


    def flip_text(x):
        return x[::-1]


    demo = gr.Blocks()

    with demo:
        gr.Markdown(
            """
        # Flip Text!
        Start typing below to see the output.
        """
        )
        input = gr.Textbox(placeholder="Flip this text")
        output = gr.Textbox()
```

```
[ ]     input.change(fn=flip_text, inputs=input, outputs=output)

    demo.launch()
```

```
[ ] import numpy as np
    import gradio as gr

    demo = gr.Blocks()


    def flip_text(x):
        return x[::-1]


    def flip_image(x):
        return np.fliplr(x)


    with demo:
        gr.Markdown("Flip text or image files using this demo.")
        with gr.Tabs():
            with gr.TabItem("Flip Text"):
                with gr.Row():
                    text_input = gr.Textbox()
                    text_output = gr.Textbox()
                text_button = gr.Button("Flip")
            with gr.TabItem("Flip Image"):
                with gr.Row():
                    image_input = gr.Image()
                    image_output = gr.Image()
                image_button = gr.Button("Flip")

        text_button.click(flip_text, inputs=text_input, outputs=text_output)
        image_button.click(flip_image, inputs=image_input, outputs=image_output)
```

```
[ ]  demo.launch()
```

```
[ ]  import gradio as gr

     api = gr.Interface.load("huggingface/EleutherAI/gpt-j-6B")


     def complete_with_gpt(text):
         # Use the last 50 characters of the text as context
         return text[:-50] + api(text[-50:])


     with gr.Blocks() as demo:
         textbox = gr.Textbox(placeholder="Type here and press enter...", lines=4)
         btn = gr.Button("Generate")

         btn.click(complete_with_gpt, textbox, textbox)

     demo.launch()
```

```
[ ]  from transformers import pipeline

     import gradio as gr

     asr = pipeline("automatic-speech-recognition", "facebook/wav2vec2-base-960h")
     classifier = pipeline("text-classification")


     def speech_to_text(speech):
         text = asr(speech)["text"]
         return text


     def text_to_sentiment(text):
         return classifier(text)[0]["label"]
```

```
[ ]     return classifier(text)[0]["label"]


    demo = gr.Blocks()

    with demo:
        audio_file = gr.Audio(type="filepath")
        text = gr.Textbox()
        label = gr.Label()

        b1 = gr.Button("Recognize Speech")
        b2 = gr.Button("Classify Sentiment")

        b1.click(speech_to_text, inputs=audio_file, outputs=text)
        b2.click(text_to_sentiment, inputs=text, outputs=label)

    demo.launch()
```

```
[ ] import gradio as gr


    def change_textbox(choice):
        if choice == "short":
            return gr.Textbox.update(lines=2, visible=True)
        elif choice == "long":
            return gr.Textbox.update(lines=8, visible=True)
        else:
            return gr.Textbox.update(visible=False)


    with gr.Blocks() as block:
        radio = gr.Radio(
            ["short", "long", "none"], label="What kind of essay would you like to write?"
        )
        text = gr.Textbox(lines=2, interactive=True)
radio.change(fn=change_textbox, inputs=radio, outputs=text)
block.launch()
```