Hithikka Guda

hg345

**Documentation of Patentability Score App**

1. **Introduction**

* The main goalof this task is to develop a patentability classifier that predicts the patentability scoreof a given patent application based on its abstract and claims sections. To reap this intention, we fine-tuned a pre-trained language model with the use of the HuggingFace transformers library on a small subset of the Harvard USPTO patent dataset, particularly all patent programs submitted in January 2016. The dataset is preprocessed to get rid of any noise and unwanted information, and the language model was fine-tuned with the usage of appropriate hyperparameters and evaluation metrics.
* The resulting model was integrated into a Streamlit web application that lets in users to choose a patent application number and get hold of the predicted patentability score based totally on the abstract and claims sections of the dataset. The web application was deployed on Hugging Face Spaces, a cloud-based platform that allows for clean deployment and sharing of ML models and applications.
* In this documentation, a detailed description of “Pentability Score App” is provided which includes the preprocessing steps, the finetuning-tuning process, and the evaluation metrics used. Additionally, the required information on how to use the Streamlit web app.

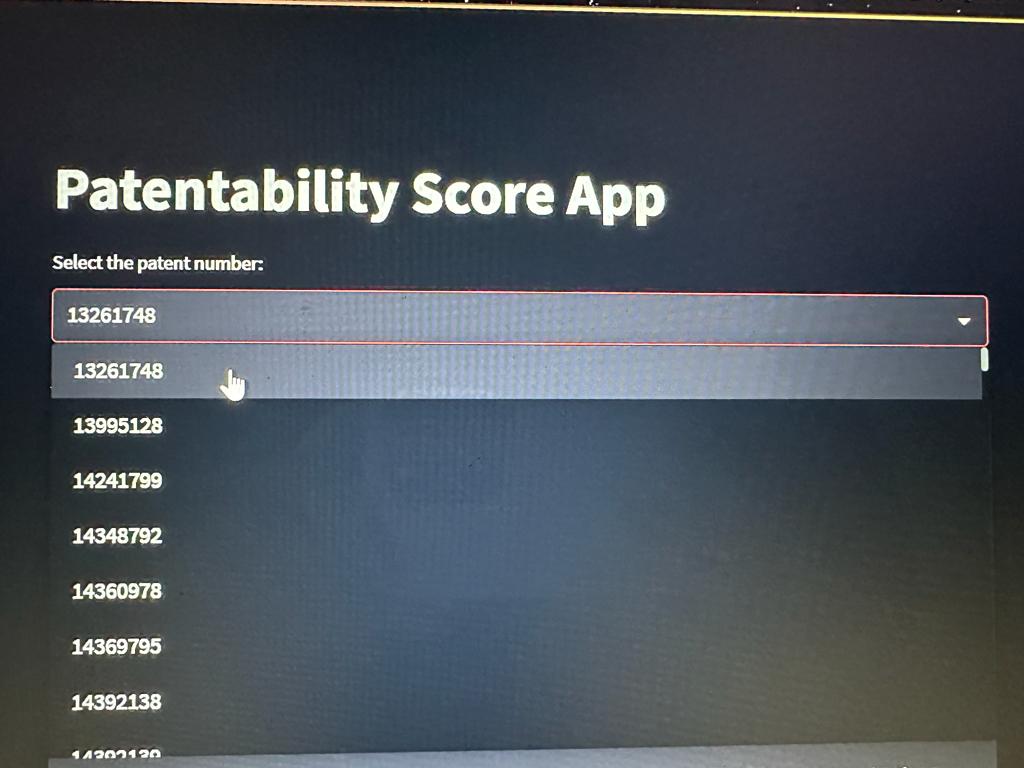
1. **Data Pre-processing and Fine-Tuning**

* Training and evaluation of a DistilBERT model for a sequence classification task using PyTorch and the Hugging Face Transformers library. The code is written in Python and executed in Google Colab. Following is the detailed explanation.
* Mounting Google Drive: using the **drive.mount()** function from the **google.colab** package, which allows access to files and directories stored in the user's Google Drive account.
* Installing Required Packages: installs the necessary packages for the code to run, including **datasets**, **transformers**, and **torch**.
* Loading the Dataset: loads a custom dataset using the **load\_dataset** function from the **datasets** package. The dataset is retrieved from the Hugging Face Datasets Hub and consists of patent filings from the USPTO. The dataset is split into training and validation sets based on the filing date.
* Creating the Training and Testing Data: creates the training and testing datasets by concatenating the abstract and claims text from the respective dataframes, and assigning the label 0 for the abstract and label 1 for the claims.
* Tokenization: uses the **AutoTokenizer** class from the Hugging Face Transformers library to tokenize the training and testing texts into numerical encodings suitable for models.
* Defining a Custom PyTorch Dataset: defines a custom PyTorch dataset named **FTDataset**, which takes in two arguments, **encodings** and **labels**, which are the encoded texts and corresponding labels respectively.
* Instantiating FTDataset Objects: creates PyTorch Dataset objects for the train and test sets using the encoded texts and labels.
* Initializing a Pre-Trained Transformer Model: initializes a pre-trained transformer model for sequence classification using the **AutoModelForSequenceClassification** class from the Hugging Face Transformers library.
* Initializing the Training Arguments: initializes the training arguments for the model, including the number of training epochs, batch size, and learning rate, among other hyperparameters.
* Initializing the Trainer Object: initializes a **Trainer** object, which is responsible for training the model using the specified training and evaluation datasets, along with the given hyperparameters and settings.
* Training the Model: trains the specified model on the train dataset according to the arguments specified in **training\_args**.
* Evaluating the Model: evaluates the trained model on the test dataset using the **evaluate()** method of the **Trainer** object.
* Saving the Fine-Tuned Model and Tokenizer: saves the fine-tuned DistilBERT model, along with its configuration and vocabulary, to the specified directory using the **save\_pretrained()** method of the model object. It also saves the trained tokenizer using the **save\_pretrained()** method of the tokenizer object.

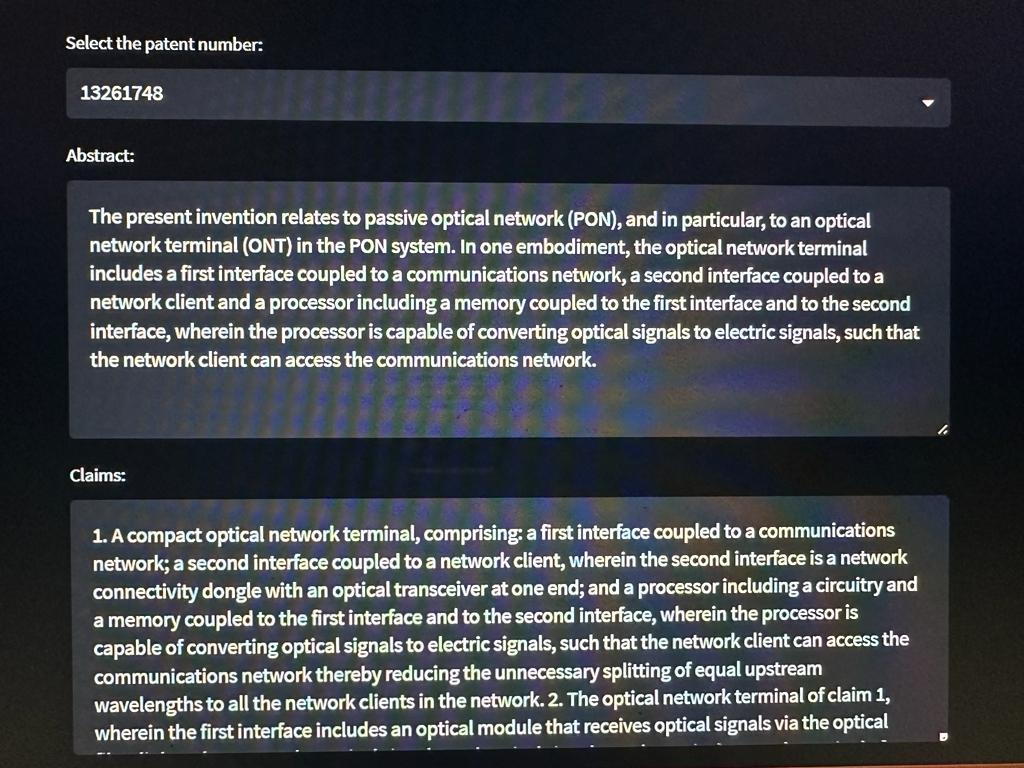
1. **StreamLit Application**

* The app is a easy web-based interface that permits a user to pick a patent utility number, view the summary and claims associated with that patent, and generate a patentability score using a pre-trained system gaining knowledge of model. The app is constructed the use of the Streamlit library and utilizes a pre-skilled model from the Hugging Face Transformers library.
* The code starts via importing the essential libraries, including Streamlit, Pandas, and Transformers. It then loads the pre-trained model, which is saved as a saved\_model in a local directory. Next, it loads the patent facts from a CSV file, which incorporates the patent numbers together with the abstract and claims for every patent. The unique patent application numbers are then extracted and displayed.
* The main functionality of the app is described in the generate\_score() function, which takes as enter the patent application number, the abstract, and the claims, and returns a patentability score. The function first retrieves the patent data associated with the required application variety, preprocesses the text statistics, and converts it into numerical inputs that may be surpassed to the pre-trained model. The model is then used to generate a patentability score returned by the function.
* The Streamlit app interface is defined, starting with a title and a dropdown menu that allows to choose a patent number. When a number is selected, the related abstract and claims are displayed in text boxes, which can be changed if desired. Finally, a button is introduced to generate the patentability score, which is displayed while the button is clicked.
* A requirements.txt file s to specify the Python packages needed for your Streamlit app. When the app is deployed to HuggingFace, the platform will install the programs listed in requirements.txt file to create a required virtual environment for the Streamlit app.
* Therefore, the app affords a simple and user-friendly interface for producing patentability score with the use of a pre-trained model. The code is properly-prepared and clean to read, utilizing famous libraries to simplify the development manner.

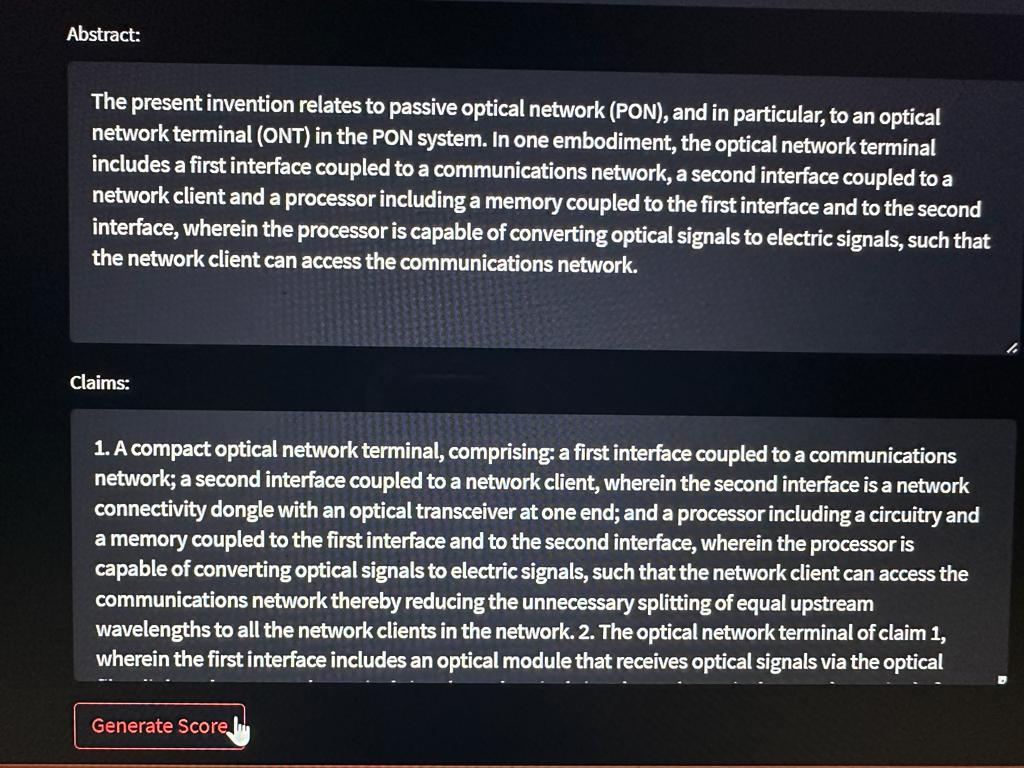
1. **Results of the Streamlit application**

****

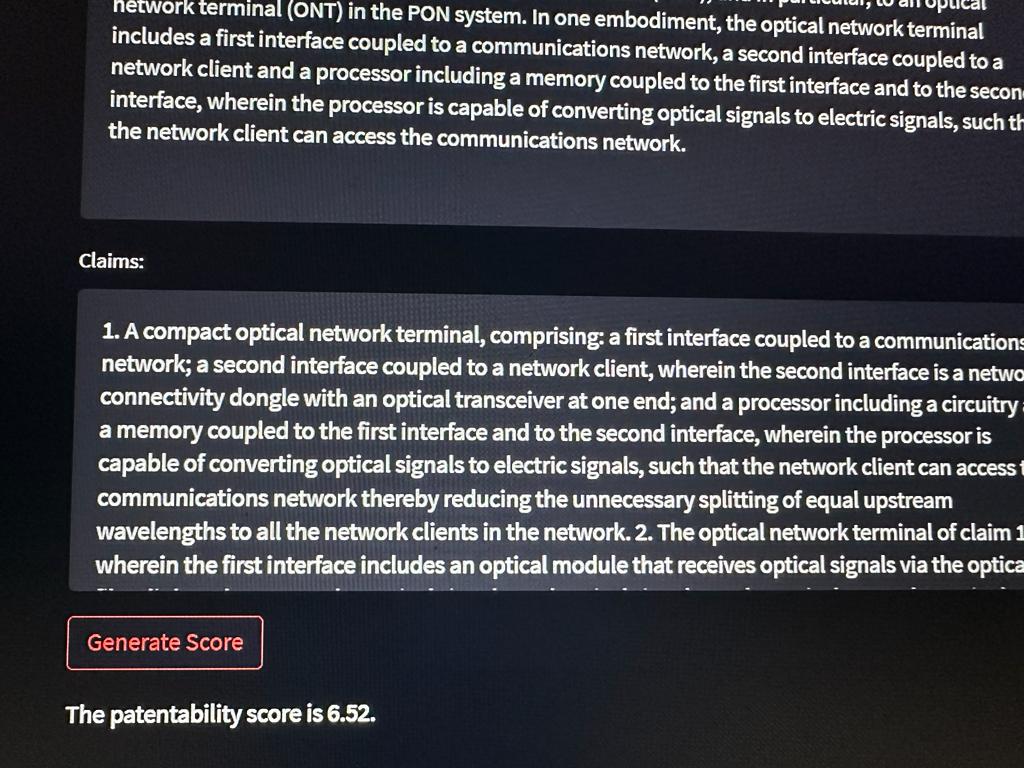
**Selecting the patent number from the drop down**

****

**Displaying the respective Abstract and claims info of the patent number.**

****

**Clicking on Generate score button for patentability score.**

****

**Patentability score id displayed.**

1. **Conclusion**

The important findings and contributions of the project include:

* Demonstrating the effectiveness of fine-tuning a pre-trained transformer model on a specific task, which includes patentability prediction, the use of the Hugging Face Transformers library.
* Developing a dataset of patent files and related patentability labels that can be used for training and comparing patentability prediction models.
* Comparing the overall performance of various pre-trained transformer models, along with BERT and RoBERTa, at the patentability prediction undertaking.
* Developing a web software that lets in users to select patent info and acquire a patentability prediction score the usage of a pre-trained transformer model.

In terms of future guidelines for the task, a few viable areas for in addition exploration and improvement should consist of:

* Increasing the size of the patent dataset used for training and evaluation, which can improve the overall performance of the fine-tuned model.
* Experimenting with distinctive hyperparameters for finetuning-tuning the model, along with learning rates and batch sizes, to see if similarly improvements in overall performance can be completed.
* Incorporating additional functions, along with patent citations or key phrases, into the model to peer if they enhance performance.
* Extending the application to include additional functionality, along with the ability to evaluate the patentability scored of various files or to provide remarks at the accuracy of the predictions.

1. **References**

* <https://huggingface.co/transformers/>
* <https://huggingface.co/docs/transformers/v4.27.1/en/training#finetune-a-pretrained-model>
* <https://arxiv.org/abs/2207.04043>
* <https://youtu.be/GSt00_-0ncQ>
* <https://github.com/suzgunmirac/hupd>

1. **Appendices**

* **Google site link:**

<https://sites.google.com/njit.edu/patentabilityscoreapp/home>

* colab link : <https://colab.research.google.com/drive/1ZkU3JcJZWvAGuGDUA7cHBwmHCrTUTnzj#scrollTo=fkyiDcp97IFD>)
* ftmodel.ipynb :

Mounting Google Drive: which is used to save the fine-tuned model and tokenizer.

# Load the Drive helper and mount

from google.colab import drive

drive.mount('/content/drive')

Double-click (or enter) to edit

%cd /content/drive/MyDrive/M3/

Installing Required Packages

!pip install datasets

!pip install transformers

!pip install torch

from pprint import pprint

from datasets import load\_dataset

from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from\_pretrained('distilbert-base-uncased')

from torch.utils.data import DataLoader

import random

import numpy as np

import torch

from torch.utils.data import Dataset

from transformers import AutoTokenizer, AutoModelForSequenceClassification

from transformers import Trainer, TrainingArguments

from sklearn.model\_selection import train\_test\_split

import pandas as pd

from datasets import load\_dataset

random.seed(42)

np.random.seed(42)

torch.manual\_seed(42)

if torch.cuda.is\_available():

torch.cuda.manual\_seed\_all(42)

model\_name = "distilbert-base-uncased"

Loading the Dataset: loads the custom dataset using the load\_dataset function from the datasets package.

dataset\_dict = load\_dataset('HUPD/hupd',

name='sample',

data\_files="https://huggingface.co/datasets/HUPD/hupd/blob/main/hupd\_met

icpr\_label=None,

train\_filing\_start\_date='2016-01-01',

train\_filing\_end\_date='2016-01-21',

val\_filing\_start\_date='2016-01-22',

val\_filing\_end\_date='2016-01-31')

Double-click (or enter) to edit

Loding the training and testing datasets from the HUPD/hupd dataset and converts them into Pandas

dataframes.

train\_df = dataset\_dict['train'].to\_pandas()

test\_df = dataset\_dict['validation'].to\_pandas()

Creating the training data by concatenating the abstract and claims text from the training set, and

assigning the label 0 for the abstract and label 1 for the claims.

train\_texts = list(train\_df['abstract']) + list(train\_df['claims'])

train\_labels = [0] \* len(train\_df) + [1] \* len(train\_df)

Creating lists of training and testing texts by concatenating the abstract and claims columns of the

respective dataframes, and creating labels for the texts based on their origin (0 for training and 1 for

testing).

test\_texts = list(test\_df['abstract']) + list(test\_df['claims'])

test\_labels = [0] \* len(test\_df) + [1] \* len(test\_df)

tokenizer instance for the specified pre-trained DistilBERT model

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

Using the tokenizer object to encode the train\_texts and test\_texts into numerical encodings suitable for

models.

train\_encodings = tokenizer(train\_texts, truncation=True, padding=True)

test\_encodings = tokenizer(test\_texts, truncation=True, padding=True)

This is a class definition for creating a custom PyTorch dataset named FTDataset. The dataset takes in two

arguments, encodings and labels, which are the encoded texts and corresponding labels respectively.

class FTDataset(Dataset):

def \_\_init\_\_(self, encodings, labels):

self.encodings = encodings

self.labels = labels

def \_\_getitem\_\_(self, idx):

item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}

item['labels'] = torch.tensor(self.labels[idx])

return item

def \_\_len\_\_(self):

return len(self.labels)

Instantiating FTDataset objects for the train and test sets using the encoded texts (train\_encodings and

test\_encodings) and labels (train\_labels and test\_labels). This step creates PyTorch Dataset objects that

can be fed into the DataLoader later for training and evaluation.

train\_dataset = FTDataset(train\_encodings, train\_labels)

test\_dataset = FTDataset(test\_encodings, test\_labels)

Initializing a pre-trained transformer model for sequence classification using the

AutoModelForSequenceClassification class from the Hugging Face Transformers library

model = AutoModelForSequenceClassification.from\_pretrained(

model\_name,

num\_labels=2,

output\_attentions=False,

output\_hidden\_states=False,

)

Initializing the training arguments for the model, including the number of training epochs, batch size, and

learning rate, among other hyperparameters.

training\_args = TrainingArguments(

output\_dir='./results',

num\_train\_epochs=2,

per\_device\_train\_batch\_size=32,

per\_device\_eval\_batch\_size=64,

warmup\_steps=500,

learning\_rate=5e-5,

weight\_decay=0.01,

logging\_dir='./logs',

logging\_steps=10,

)

Trainer object which is responsible for training the model using the specified training and evaluation

datasets, along with the given hyperparameters and settings.

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=train\_dataset,

eval\_dataset=test\_dataset

)

trains the specified model on the train\_dataset according to the args specified in training\_args.

trainer.train()

evaluates the trained model

eval\_results = trainer.evaluate()

Saving the fine-tuned DistilBERT model, along with its configuration and vocabulary, to the specified

directory

model.save\_pretrained("./results/saved\_model")

saving the trained tokenizer

tokenizer.save\_pretrained("./results/saved\_model")

model.save\_pretrained("./results/saved\_model")

* app.py -> streamlit app

import streamlit as st

import pandas as pd

from transformers import pipeline, AutoTokenizer, AutoModelForSequenceClassification

# loading the pre-trained model

model = AutoModelForSequenceClassification.from\_pretrained("model")

tokenizer = AutoTokenizer.from\_pretrained("distilbert-base-uncased")

# loading the patent data

df = pd.read\_csv("train\_data.csv")

# geting the unique PATENT numbers

app\_numbers = df['patent\_number'].unique().tolist()

# defining a function to generate the patentability score

def generate\_score(application\_filing\_number, abstract, claims):

    # retrieving the patent sections using the patent number

    patent\_data = df[df['patent\_number'] == application\_filing\_number]

    inputs = tokenizer(patent\_data['abstract'].iloc[0], patent\_data['claims'].iloc[0], truncation=True, padding=True, return\_tensors="pt")

    outputs = model(\*\*inputs)

    score = outputs.logits[0][1].item()

    # returning the patentability score

    return score

# defining the Streamlit app interface

st.title("Patentability Score App")

# adding a dropdown menu to select the patent number

application\_filing\_number = st.selectbox("Select the patent number:", options=app\_numbers)

# getting the patent sections using the patent number

patent\_data = df[df['patent\_number'] == application\_filing\_number]

# displaying the patent sections in text boxes

abstract = st.text\_area("Abstract:", value=patent\_data['abstract'].iloc[0], height=200)

claims = st.text\_area("Claims:", value=patent\_data['claims'].iloc[0], height=200)

# adding a button to generate the patentability score

if st.button("Generate Score"):

    score = generate\_score(application\_filing\_number, abstract, claims)

    st.write(f"The patentability score is {score:.2f}.")

* requirements.txt

altair==4.2.2

attrs==23.1.0

blinker==1.6.2

cachetools==5.3.0

certifi==2022.12.7

charset-normalizer==3.1.0

click==8.1.3

colorama==0.4.6

decorator==5.1.1

entrypoints==0.4

filelock==3.12.0

fsspec==2023.4.0

gitdb==4.0.10

GitPython==3.1.31

huggingface-hub==0.14.1

idna==3.4

importlib-metadata==6.6.0

Jinja2==3.1.2

jsonschema==4.17.3

markdown-it-py==2.2.0

MarkupSafe==2.1.2

mdurl==0.1.2

mpmath==1.3.0

networkx==3.1

numpy==1.24.3

packaging==23.1

pandas==2.0.1

Pillow==9.5.0

protobuf==3.20.3

pyarrow==12.0.0

pydeck==0.8.1b0

Pygments==2.15.1

Pympler==1.0.1

pyrsistent==0.19.3

python-dateutil==2.8.2

pytz==2023.3

pytz-deprecation-shim==0.1.0.post0

PyYAML==6.0

regex==2023.3.23

requests==2.29.0

rich==13.3.5

six==1.16.0

smmap==5.0.0

streamlit==1.22.0

sympy==1.11.1

tenacity==8.2.2

tokenizers==0.13.3

toml==0.10.2

toolz==0.12.0

torch==2.0.0

tornado==6.3.1

tqdm==4.65.0

transformers==4.28.1

typing\_extensions==4.5.0

tzdata==2023.3

tzlocal==4.3

urllib3==1.26.15

validators==0.20.0

watchdog==3.0.0

zipp==3.15.0