[Fine Tune] Fine Tuning T5 for Text Generation

Shawn Ding · Follow
5 min read · Jan 5

T5 is a state-of-the-art language model developed by Google Research that can perform various NLP tasks, such as translation, summarization, and text generation. In this blog, we will explore how to fine-tune T5 for text generation and demonstrate the results with a few examples. We will also discuss some best practices and considerations when fine-tuning T5 for text generation.

This is a comprehensive tutorial for text generation.

✍️ Dataset💦

We will use the Extreme Summarization (XSum) Dataset. The dataset consists of BBC articles and accompanying single sentence summaries. Specifically, each article is prefaced with an introductory sentence (aka summary) which is professionally written, typically by the author of the article. There are two features in this dataset: (1) document: Input news article. (2) summary: One sentence summary of the article. The idea is to generate a short, one-sentence news summary answering the question "What is the article about?". There are in total 226k samples: 204,045 samples for training data, 11,332 samples for validation data and 11,334 samples for test data. The average number of words in a document is 431.07 (19.77 sentences) and the average number of words in a summary is 23.26. Evaluation Metric For this task, we will use ROUGE-2 (Recall-Oriented Understudy for Gisting Evaluation) for model evaluation.

XSum Dataset: We will use python lib 'datasets' to load the XSum, for example: train_data = datasets.load_dataset("xsum", split="train"). You will see the specific code at **Step. 2**

✍️ STEP.1 IMPORT PYTHON LIBRARY 💦

```
# !pip install transformers -q
# !pip install datasets -q
# !pip install rouge -q

import json
import pandas as pd
import numpy as np
import torch
import pytorch as pl

import re
import string
import operator
import numpy as np
import random

from torch.utils.data import Dataset, DataLoader
from torch import ModelCheckpoint
from transformers import AdamW
from transformers import T5Tokenizer
from transformers import T5ForConditionalGeneration

seed = 38
device = torch.device('cuda')
```

✍️ STEP.2 Read the dataset💦

```
import datasets

train_data = datasets.load_dataset("xsum", split="train")
val_data = datasets.load_dataset("xsum", split="validation")
test_data = datasets.load_dataset("xsum", split="test")
print(len(train_data['document']),len(train_data['summary']))
```

✍️ STEP.3 Pre-Processing of the dataset💦

We imports the `Series` and `DataFrame` classes from the `pandas` library. It then checks the lengths of the 'document' and 'summary' columns in the `train_data` dataframe and assigns them to variables `len(train_data['document'])` and `len(train_data['summary'])`.

Next, the code creates a new dictionary called `data` with the keys 'document' and 'summary' and values `train_data['document']` and `train_data['summary']`, respectively. This `data` dictionary is then used to create a new `DataFrame` called `train_df`. The shape of `train_df` is then printed to the console.

The same process is repeated to create `val_df` and `test_df` dataframes using the `val_data` and `test_data` dictionaries, respectively. The shape of these dataframes is also printed to the console.

```
from pandas import Series,DataFrame
len(train_data['document']),len(train_data['summary'])
data = {'document':train_data['document'],
       'summary':train_data['summary']}
train_df = DataFrame(data)
train_df.shape

data = {'document':val_data['document'],
       'summary':val_data['summary']}
val_df = DataFrame(data)
val_df.shape

data = {'document':test_data['document'],
       'summary':test_data['summary']}
test_df = DataFrame(data)
test_df.shape
```

✍️ STEP.4 Construction of some python function💦

We define several functions related to data loading, encoding, and training a machine learning model for text generation.

The `load_data` function takes in a `train_dataset` argument and returns a data loader with a batch size of 8 and shuffle set to True. The `load_data_` function is similar to `load_data`, but the shuffle argument is not set, so the data is not shuffled.

The `encoding_process` function takes in a `doc`, `doc_lst`, and `sum_lst` as arguments and returns several variables including `document`, `summary`, `doc_input_ids`, `doc_attention_mask`, and `labels_attention_mask`. This function appears to encode the `doc` and `summary` using the `tokenizer` function and returns these encodings as well as the original `document` and `summary` lists.

```
torch.cuda.empty_cache()
pl.seed_everything(36)

def load_data(train_dataset):
    DataLoader(train_dataset,
        # batch_size=6,
        batch_size=8,
        shuffle=True,
    )

def load_data_(test_dataset):
    DataLoader(test_dataset,
        # batch_size=6,
        batch_size=8,
    )

def encoding_process(doc, doc_lst,sum_lst):
  doc_lst = []
  for j in df['document']:
    doc_lst.append(j)
  sum_lst=[]
  for j in df['summary']:
    sum_lst.append(j)
  doc_encode = tokenizer(doc,max_length=self.text_max_token_len,padding='max_leng
  sum_encode = tokenizer(data_row['summary'], max_length=self.summary_max_token_le
  labels = summary_encoding['input_ids']
  labels=labels.flatten()

  doc_input_ids=doc_encode['input_ids']
  doc_attention_mask=doc_encode['attention_mask']
  labels_attention_mask=sum_encode['attention_mask']

  document=doc_lst
  summary=sum_lst

return document, summary, doc_input_ids, doc_attention_mask, labels_attention_mas

doc_train = load_train(df_train)
doc_val = load_data_(df_val)
doc_test = load_data_(doc_test)

encoding_process(df_train,doc_lst,sum_lst)
encoding_process(df_va,val_doc_lst,val_sum_lstl)
```

```
def prediction(doc, word_lst):
    document_encoding = tokenizer(
      doc,
      max_length=512,
      padding='max_length',
      truncation=True,
      return_attention_mask=True,
      add_special_tokens=True,
      return_tensors='pt'
    )

    pre_word = model.generate(
      input_ids=document_encoding['input_ids'],
      attention_mask=document_encoding['attention_mask'],
      max_length=256,
      num_beams=1,
    )
    word_lst = []
    for i in pre_word:
      word_lst.append(i)
      str1 = "".join(i)
      str2 = "".join(i)
      str3 = "".join(word_lst)
    return str3
```

The `prediction` function takes in a `doc` and `word_lst` as arguments and returns a predicted summary as a string. It appears to use the `model.generate` method to generate a summary from the encoded `doc` .

The `fn` function takes in several arguments including `model` , `input_ids` , `attention_mask` , and `decoder_attention_mask` , and returns the `output` and `loss` of the model.

The `training_valid` function appears to be used for training and validation of the model. It takes in several arguments including `text_encoding` , `batch_size` , `val_batch` , `val_size` , and `model` . It appears to split the input data into training and validation sets and then trains and validates the model on these sets. It returns several variables including `epoch_loss` , `val_loss` , `val_acc` , and `val_f1` .

```
def fn(model, input_ids, attention_mask, decoder_attention_mask):
  output = model(
      train_input_ids,
      train_attention_mask=attention_mask,
      train_labels=labels,
      train_decoder_attention_mask=decoder_attention_mask)
  return output, output.loss

def training_valid(text_encoding, batch_size,val_batch,val_size,model = pl.model)
  train_input_ids = text_encoding['text_input_ids']
  train_labels = text_encoding['labels']
  val_input_ids = val_text_encoding['text_input_ids']
  val_labels = val_text_encoding['labels']
  train_attention_mask = text_encoding['text_attention_mask']
  train_labels_attention_mask = text_encoding['labels_attention_mask']
  val_attention_mask = val_text_encodingh['text_attention_mask']
  val_labels_attention_mask = val_text_encoding['labels_attention_mask']

  train_loss, train_outputs = model(
      train_input_ids=input_ids,
      train_attention_mask=attention_mask,
      train_decoder_attention_mask=labels_attention_mask,
      train_labels=labels
  )

  val_loss, val_outputs = model(
      val_input_ids=val_input_ids,
      val_attention_mask=val_attention_mask,
      val_decoder_attention_mask=val_labels_attention_mask,
      val_labels=val_labels
  )
  # print("training_loss", loss)
  # print("valid_loss", val_loss)
  return train_loss, val_loss

output_model = './model/myT5.pth'
# save
#optimizer
def save(model,op):
  # save
  torch.save({
      'model_state_dict': model.state_dict(),
      'op': op.state_dict()
  }, output_model)
```

✍️ STEP.5 Fine-tuning T5 for our dataset 💦

The `model` is re-defined as an instance of the T5ForConditionalGeneration class, with the `'t5-small'` pre-trained model and `return_dict=True` as arguments. The `train_data` and `val_data` are then loaded using the `load_train` and `load_val` functions, respectively, and passed to the `trainer.fit` method along with the `model`. The trained model and optimizer are then saved using the `save` function.

😉 Please remember to create the path in your local: '/model/'

```
# t5-small 63M params
fn(model, doc_train_input, doc_m, decoder_m)
model = T5ForConditionalGeneration.from_pretrained('t5-small', return_dict=True)
tokenizer = T5Tokenizer.from_pretrained('t5-small')
N_EPOCHS = 2
# 4 , 8
BATCH_SIZE = 6
lr=0.0002

model = T5ForConditionalGeneration.from_pretrained('t5-small', return_dict=True)
train_data = load_train(df_train)
val_data = load_val(df_val)
# train_data = load_train(df_test)
trainer.fit(model, doc_train, doc_val)
save(model, optimizer)
```

In this blog, we have explored how to fine-tune the T5 model for text generation using the XSum dataset. We have discussed the steps for data pre-processing, construction of python functions, and fine-tuning XLNet for our dataset. We have also seen the results of fine-tuning the T5 model for text generation, and discussed some best practices and considerations. Finally, we have saved the trained model and optimizer.

If you want the full code, please let me know! 👏

All materials in this tutorial refer to: https://github.com/xding2/Fine-Tuning-NLP-Model 👏

NLP Deep Learning Python T5 Text Generation

Follow

Written by Shawn Ding

187 Followers

Hello everyone! I am the NLP researcher working at SAIL.

More from Shawn Ding

Shawn Ding

[Fine Tune] Fine Tuning BERT for Sentiment Analysis

Key tags: fine-tune; BERT; sentiment analysis; classification

8 min read · Jan 2

Shawn Ding

[Fine Tune] Fine Tuning XLNet for Relation Extraction.

Fine-tuning a pre-trained transformer model, such as XLNet, on a specific task involves adapting the model's weights to the new task by...

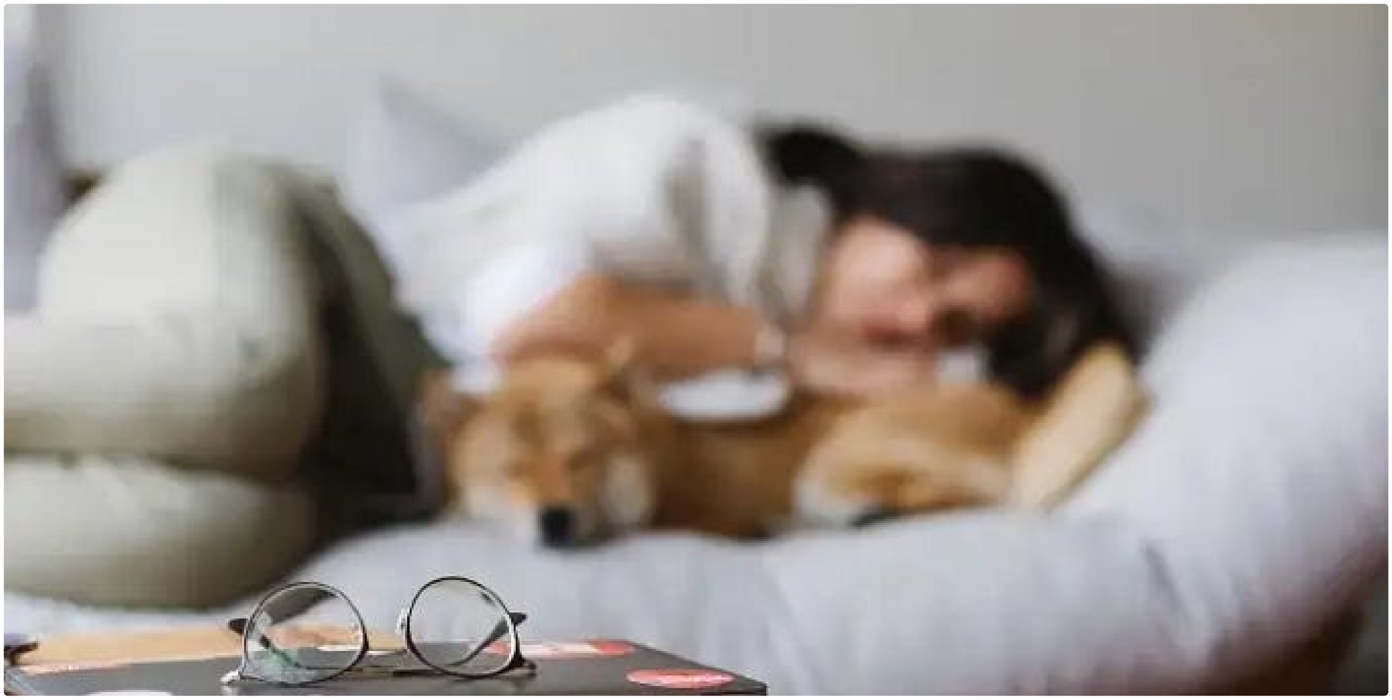
8 min read · Jan 3

Shawn Ding

[Fine Tune] Fine Tuning BERT for Question Answering (QA) Task

Question Answering (QA) is a type of natural language processing task where a model is trained to answer questions based on a given context…

4 min read · Jan 22



Shawn Ding

GPT-2 from Scratch: ‘Les Misérables’ Written by GPT to the Surprise of All!

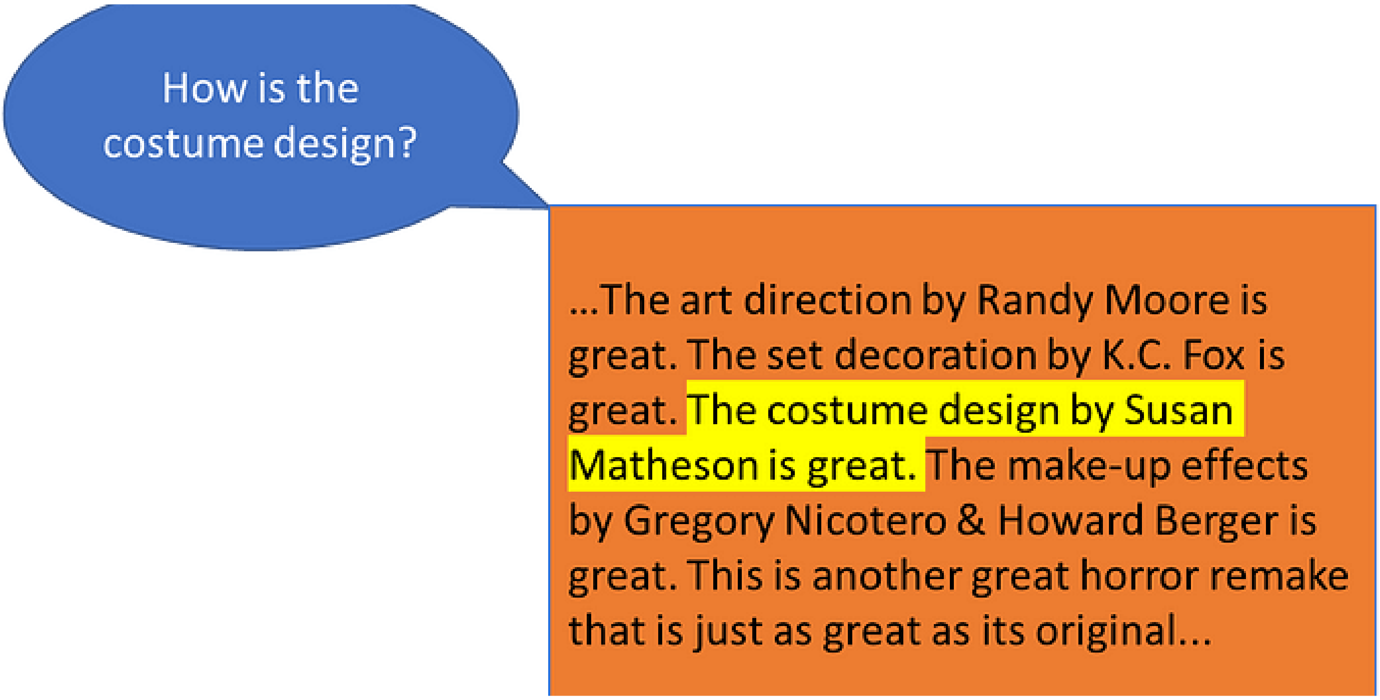
GPT-2, a state-of-the-art natural language processing model developed by OpenAI, has caused a stir in the artificial intelligence community...

7 min read · Jan 13

5 1

See all from Shawn Ding

Recommended from Medium

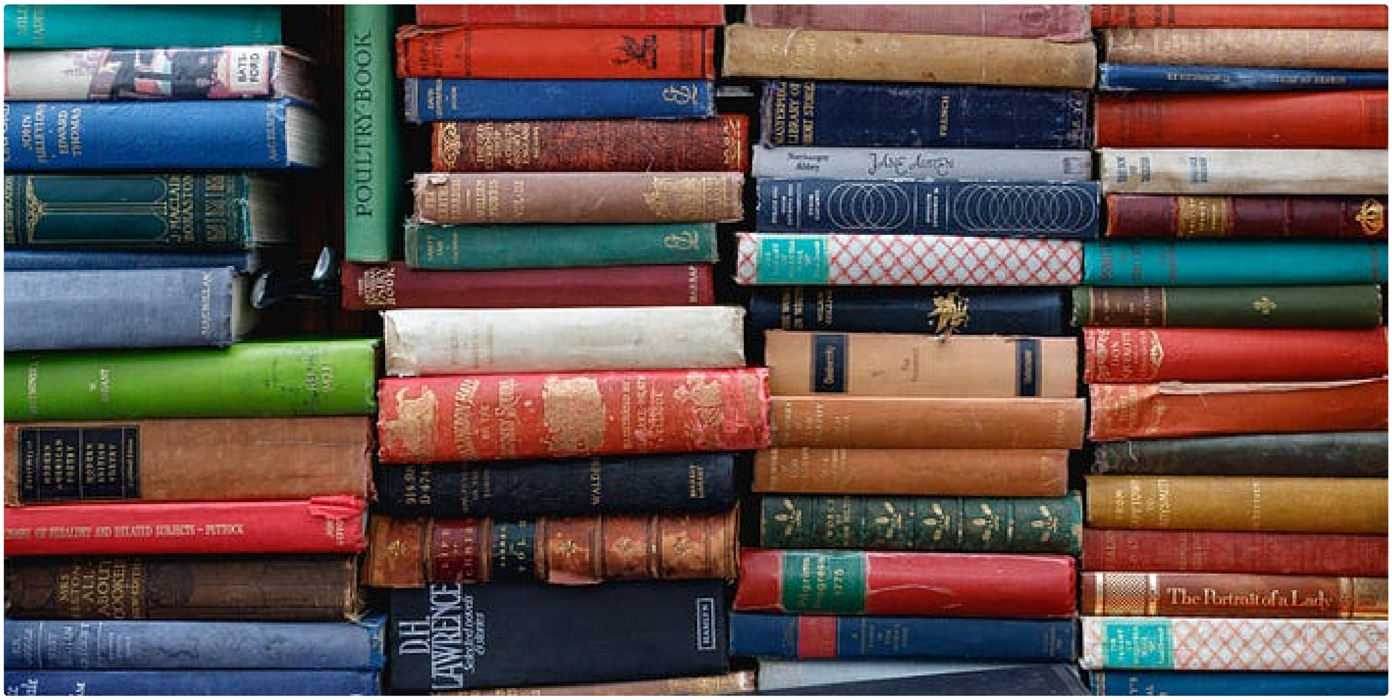


Skanda Vivek in Towards Data Science

Fine-Tune Transformer Models For Question Answering On Custom Data

A tutorial on fine-tuning the Hugging Face RoBERTa QA Model on custom data and obtaining significant performance boosts

· 5 min read · Dec 15, 2022



Jay Peterman in Towards Data Science

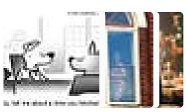
Make a Text Summarizer with GPT-3

Quick tutorial using Python, OpenAI's GPT-3, and Streamlit

· 11 min read · Jan 24

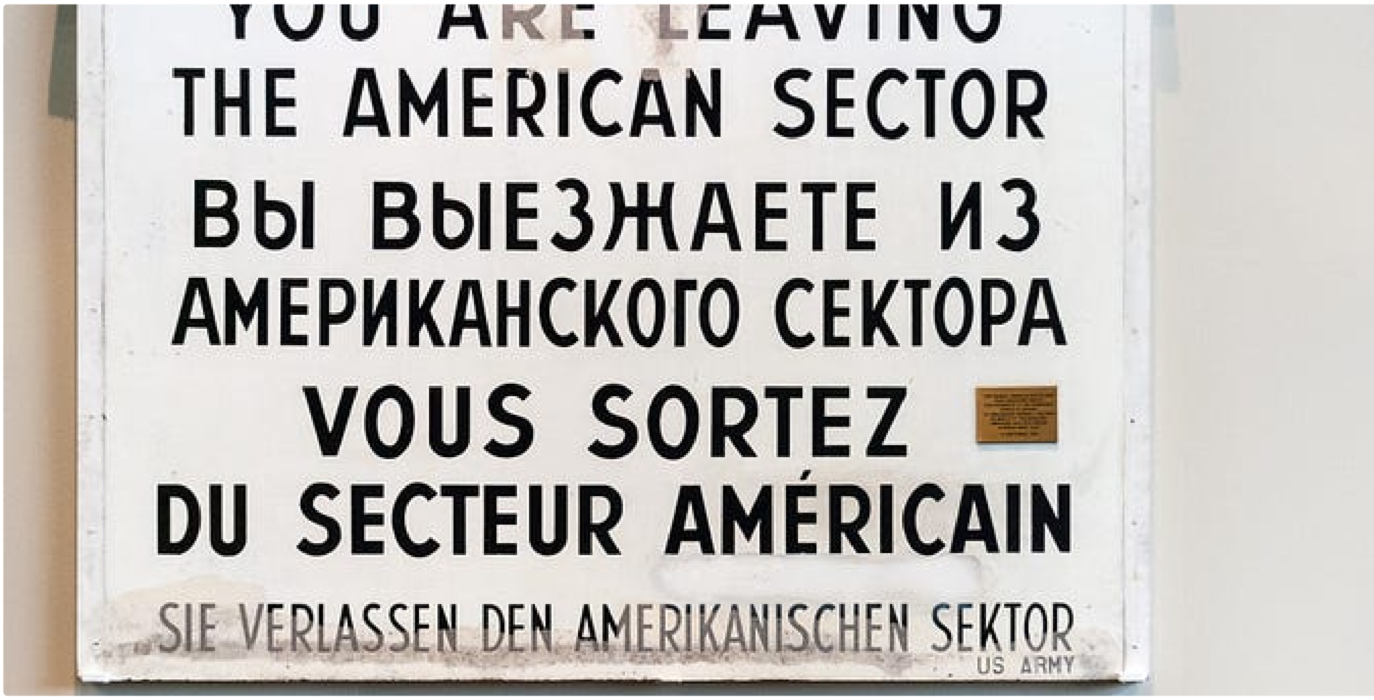
153 1

Lists



Staff Picks

339 stories · 95 saves



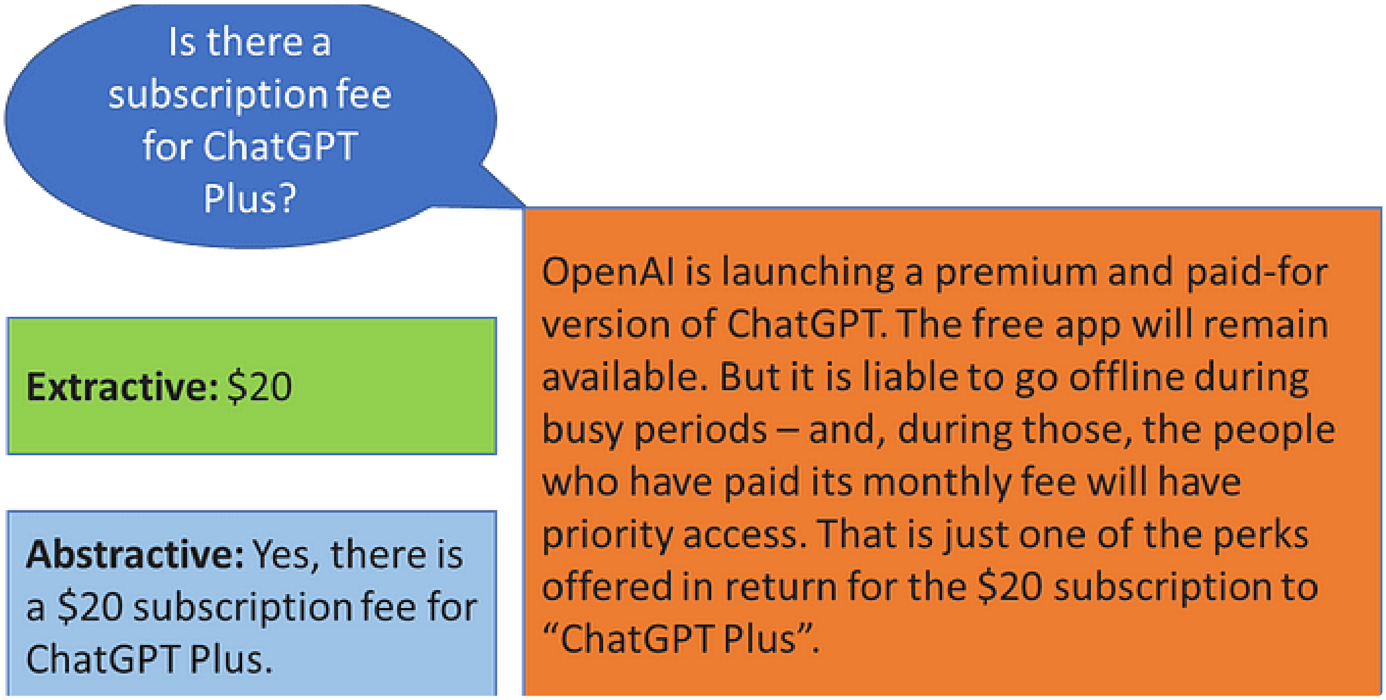
Ala Alam Falaki in Towards AI

Fine-Tune BART for Translation on WMT16 Dataset (and Train new Tokenizer)

BART is a well-known summarization model. Also, it can do the translation task with the appropriate tokenizer for the target language.

· 5 min read · Mar 13

1 1

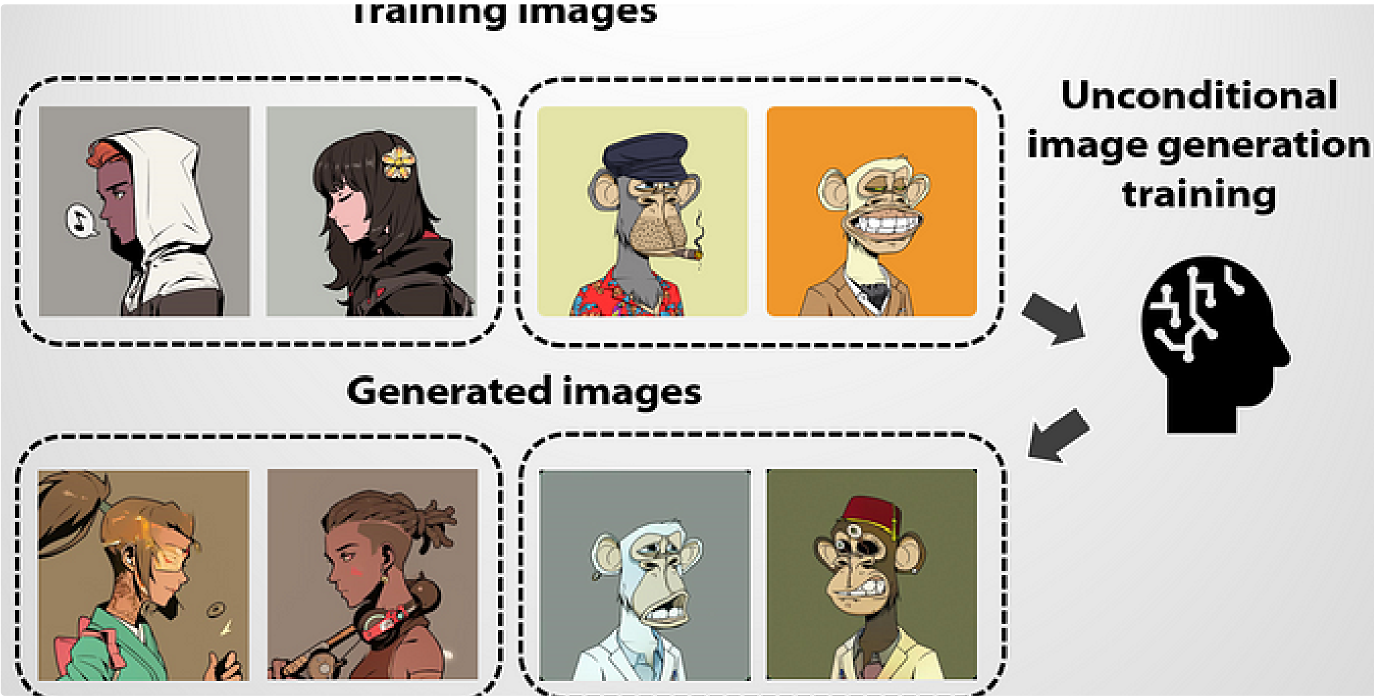


Skanda Vivek in Towards Data Science

Extractive vs Generative Q&A — Which is better for your business?

The arrival of ChatGPT hints at a new era of search engines, this tutorial dives into the 2 basic types of AI based question answering

· 6 min read · Feb 6



Ng Wai Foong in Better Programming

The Beginner's Guide to Unconditional Image Generation Using Diffusers

Explore and generate unique and imaginative images based on existing datasets

· 8 min read · Feb 1

74 1

Dr. Mandar Karhade, MD. PhD. in Geek Culture

Page by Page Review: LLaMA: Open and Efficient Foundation Language Model

LLaMA-13B outperforms GPT-3 (175B) in most benchmarks, and LLaMA-65B can even do your dirty laundry!

· 7 min read · Mar 2

88 2

See more recommendations