**Deploying an NLP App with Streamlit and Hugging Face**

Step-by-step guide to showcase your models online (for free)

[[Ivan Lee](https://ivan-lee.medium.com/?source=post_page-----453b90e58c5a--------------------------------)](https://ivan-lee.medium.com/?source=post_page-----453b90e58c5a--------------------------------)

[[GoPenAI](https://blog.gopenai.com/?source=post_page-----453b90e58c5a--------------------------------)](https://blog.gopenai.com/?source=post_page-----453b90e58c5a--------------------------------)

[Ivan Lee](https://ivan-lee.medium.com/?source=post_page-----453b90e58c5a--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F4ab21a62835d&operation=register&redirect=https%3A%2F%2Fblog.gopenai.com%2Fdeploy-nlp-app-with-streamlit-and-hugging-face-453b90e58c5a&user=Ivan+Lee&userId=4ab21a62835d&source=post_page-4ab21a62835d----453b90e58c5a---------------------post_header-----------)

Published in

[GoPenAI](https://blog.gopenai.com/?source=post_page-----453b90e58c5a--------------------------------)

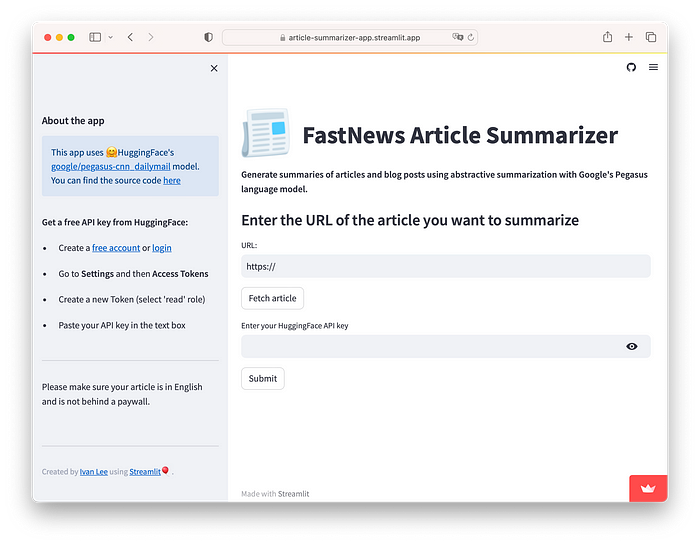
·

7 min read

·

Jul 24

64



Having access to powerful large language models from **OpenAI**, **Google** and others is now easier than ever. Leveraging these state-of-the-art models, developers can experiment and build powerful applications. One way to conveniently deploy and showcase your models on the web is with [**Streamlit**](https://streamlit.io/) (no *HTML* or *JavaScript* needed).

In this article, I will show how to create and deploy an application using Streamlit. For this project, I built a simple app that summarizes online articles or blog posts using Google’s language model **Pegasus** from the [**Hugging Face**](https://huggingface.co/) platform.

You can check the code on [**GitHub**](https://github.com/ivnlee/streamlit-text-summarizer) and try the app for yourself [**here**](https://article-summarizer-app.streamlit.app/).



**What is Streamlit?**

Streamlit is an open-source Python library that allows you to create interactive web apps with a few modifications to your existing Python script. Also, Streamlit offers a free hosting service where you can deploy your app from GitHub.

Streamlit is compatible with several libraries and frameworks such as Pandas, Matplotlib, Scikitlearn, Pytorch, TensorFlow, and more.

**Getting started**

Install the package in your environment via pip (it is recommended that you set up a virtual environment first, see the guide [here](https://docs.streamlit.io/library/get-started/installation)):

$ pip install streamlit

Later on, we will need to create a **requirements.txt** file, one way to do that is with the pipreqs package, so let’s install it in advance:

$ pip install pipreqs

Create a new Python file using the IDE of your choice and start by importing all the required libraries. Depending on your project specifics, just import the required libraries as usual.

import streamlit as st  
import json  
import requests  
import time  
from newspaper import Article

**Text elements**

Streamlit apps are just Python scripts that run from top to bottom. It has a wide variety of UI elements (called widgets), for example, this line of code will display the title of my app:

st.title("FastNews Article Summarizer")

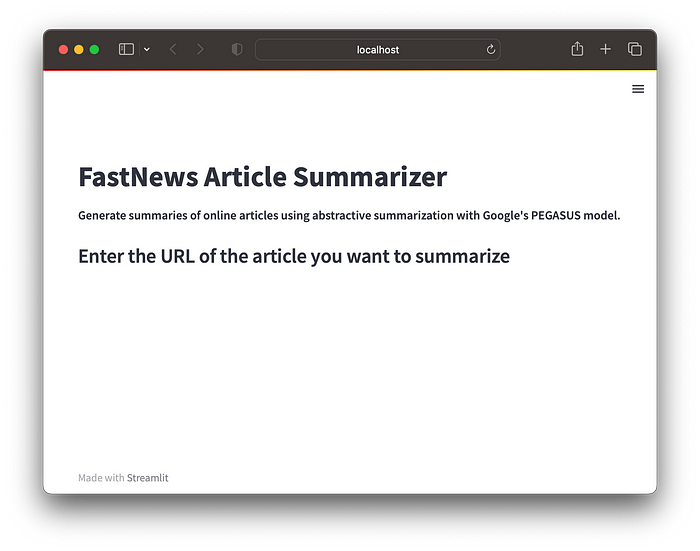
You can display text in **Header** formatting, **Sub-header** formatting and even **Markdown**:

st.markdown("\*\*Generate summaries of online articles using abstractive summarization with Google's PEGASUS model.\*\*")  
st.subheader("Enter the URL of the article you want to summarize")

To check how your app is looking, execute this command in the terminal (save any file changes before running the command):

$ streamlit run your\_app.py

A browser window will launch locally displaying your app:



Displaying Title, Markdown and Sub-header text.

**Input Elements**

My application requires the user to enter a URL. For that purpose, we can assign the widget st.text\_input() to a variable and treat it as any other variable in Python:

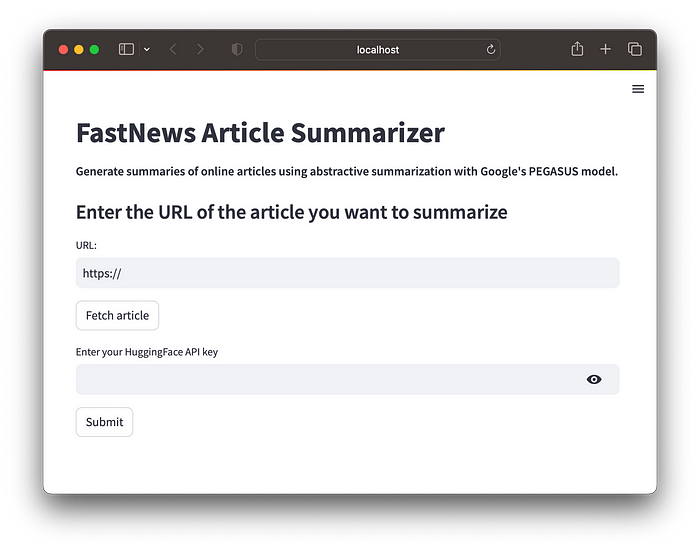
default\_url = "https://"  
url = st.text\_input("URL:", default\_url)

With st.button("Fetch article") we create a button that the user will press after entering the URL.

In addition to the article’s URL, my app requires the user to input a valid API key from **HuggingFace**, and I want to mask that text for privacy. To do that, we simply pass the argument type="password" to the st.text\_input() widget:

API\_KEY = st.text\_input("Enter your HuggingFace API key", type="password")

Once the user enters the API key, the button st.button("Submit") will start the article extraction, then post the request to the Hugging Face API endpoint, and finally display the results.



Input text fields and input buttons.

**Scraping Articles from the Web**

1. Use the requests library to scrape the content of the article from its URL.
2. Next, using the newspaper library, parse the scraped HTML and extract the text from the article.

To implement **Step 1**, the following block of code will be executed if the “Fetch article” button has been clicked:

headers\_ = {  
 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36'  
 }  
  
if fetch\_button:  
 article\_url = url  
 session = requests.Session()  
  
 try:  
 response\_ = session.get(article\_url, headers=headers\_, timeout=10)  
   
 if response\_.status\_code == 200:  
  
 with st.spinner('Fetching your article...'):  
 time.sleep(3)  
 st.success('Your article is ready for summarization!')  
  
 else:  
 st.write("Error occurred while fetching article.")  
  
 except Exception as e:  
 st.write(f"Error occurred while fetching article: {e}")

This try-except block handles exceptions that may occur while fetching the article. The requests.get method sends an HTTP GET request to the URL with the specified headers and timeout. If the response's status code is 200 (indicating a successful request), the if statement is executed. Otherwise, we use the st.write() widget to display an error message. If any exception occurs during the request, the except statement is executed, and the exception error (stored in the variable e) is displayed.

The st.spinner() widget inside the if statement creates a loading spinner with the message *“Fetching your article…”*. Finally, the st.success() widget displays the success message *“Your article is ready for summarization!”*

For **Step 2**, we use the Article class from the newspaper library:

if submit\_button:  
 article = Article(url)  
 article.download()  
 article.parse()  
  
 title = article.title  
 text = article.text

These lines extract the title and text content of the article using the attributes title and text of the article object.

**Hugging Face Inference API**

Hugging Face is an AI community that hosts open-source **ML** models, with a focus on **Transformers** for Natural Language Processing (NLP). The invention of the Transformer architecture in 2017, revolutionized the field of NLP. The transformer made possible the creation of LLMs such as **GPT**, **BERT**, **T5**, and many others.

I will be using the model **Pegasus**from the Hugging Face hub**.**Pegasus is an encoder-decoder-style transformer pre-trained for summarization. The specific checkpoint is: [**google/pegasus-cnn\_dailymail**](https://huggingface.co/google/pegasus-cnn_dailymail), fine-tuned with the [**CNN-dailymail**](https://huggingface.co/datasets/cnn_dailymail) corpus.

Running a model from the hosted inference API is pretty straightforward. I will use the free inference API endpoint:

API\_URL = "https://api-inference.huggingface.co/models/google/pegasus-cnn\_dailymail"

To make the request we define a “header”, which contains the user’s API key previously stored in the variable API\_KEY:

headers = {"Authorization": f"Bearer {API\_KEY}"}

Next, the function query will send a POST request to the API endpoint. The extracted text from the article will be sent in JSON format. After receiving the response, the function returns the JSON content using the response.json() method. This method will convert the JSON response into a Python object that can be easily accessed and manipulated.

def query(payload):  
 response = requests.post(API\_URL, headers=headers, json=payload)  
 return response.json()

Finally, we send our query passing the variable text as the argument and store the clean summarized text in the variable summary. Note that we replaced the string <n> which is our model’s specific way of representing blank space tokens:

output = query({"inputs": text, })  
  
 # Display the results  
 summary = output[0]['summary\_text'].replace('<n>', " ")   
  
 st.divider()  
 st.subheader("Summary")  
 st.write(f"Your article: \*\*{title}\*\*")  
 st.write(f"\*\*{summary}\*\*")

The summary is displayed using the st.write() widget. The text in this widget can be treated as Markdown text as well.

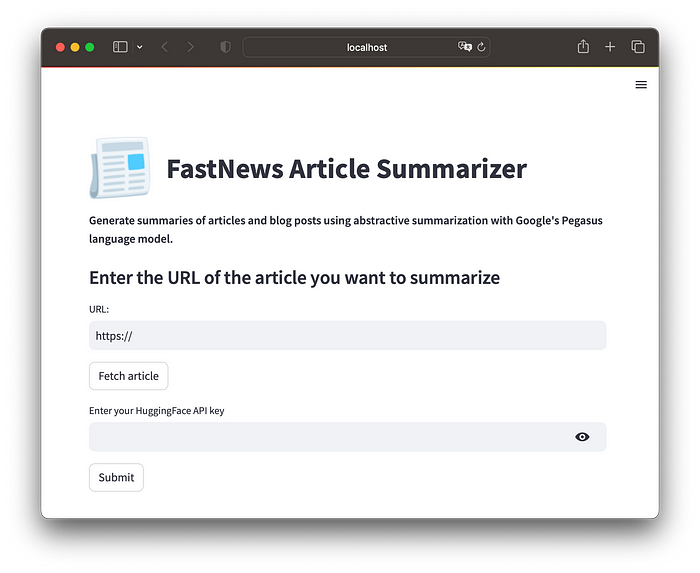
**Finishing touches**

Streamlit lets you display images, audio and video files. For example, let’s add a .png image in the title section. First, let’s create two columns:

c1, c2 = st.columns([0.32, 2])  
  
with c1:  
 st.image("images/newspaper.png", width=85)  
  
with c2:  
 st.title("FastNews Article Summarizer")

Columns c1 and c2 are created using the st.columns() widget, c1 with a width of 0.32 and c2 twice that value.

The st.image() widget requires the image’s local file path or the image URL. In this case, I also set the image width to 85 pixels.

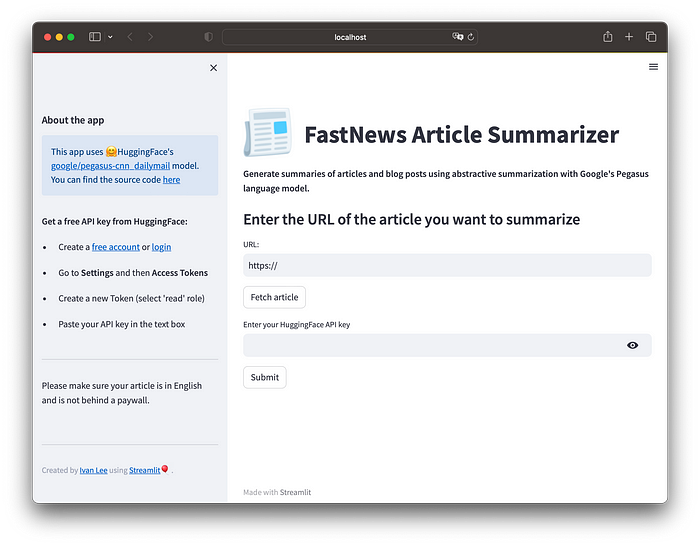


Displaying an image

Finally, let's add a sidebar section where I want to display some information about the app. This is easily done with the st.sidebar object as follows:

st.sidebar.subheader("About the app")  
st.sidebar.info("This app uses 🤗HuggingFace's [google/pegasus-cnn\_dailymail](https://huggingface.co/google/pegasus-cnn\_dailymail) model.\  
 \nYou can find the source code [here](https://github.com/ivnlee/streamlit-text-summarizer)")  
st.sidebar.write("\n\n")  
st.sidebar.markdown("\*\*Get a free API key from HuggingFace:\*\*")  
st.sidebar.markdown("\* Create a [free account](https://huggingface.co/join) or [login](https://huggingface.co/login)")  
st.sidebar.markdown("\* Go to \*\*Settings\*\* and then \*\*Access Tokens\*\*")  
st.sidebar.markdown("\* Create a new Token (select 'read' role)")  
st.sidebar.markdown("\* Paste your API key in the text box")  
st.sidebar.divider()  
st.sidebar.write("Please make sure your article is in English and is not behind a paywall.")  
st.sidebar.write("\n\n")  
st.sidebar.divider()  
st.sidebar.caption("Created by [Ivan Lee](https://ivan-lee.medium.com/) using [Streamlit](https://streamlit.io/)🎈.")

The widget st.info() displays an informational message, and st.caption() is used to display footnotes or side-note text:



Adding sidebar content

**Deployment**

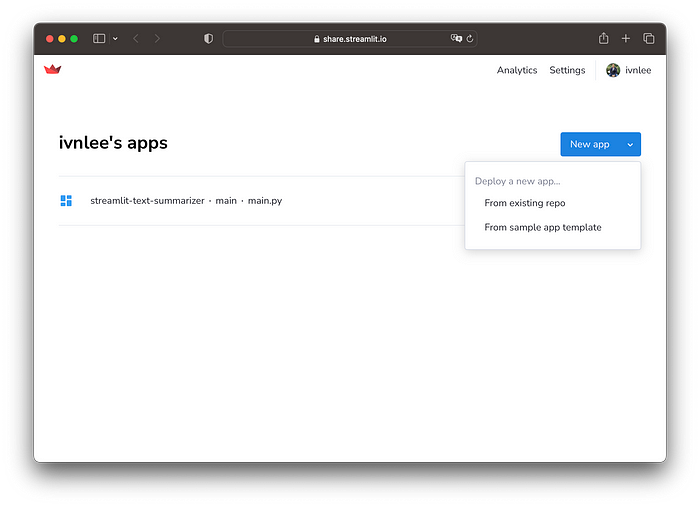
To deploy the app, first sign up to [**Streamlit Cloud**](https://share.streamlit.io/signup)(you will be asked to link your GitHub account to Streamlit). We also need to create a public GitHub repository containing all our project files.

As I mentioned earlier, we need a **requirements.txt** file so that Streamlit knows which dependencies are needed to run the app. In the terminal, navigate to the project folder and simply enter the command:

$ pipreqs ./

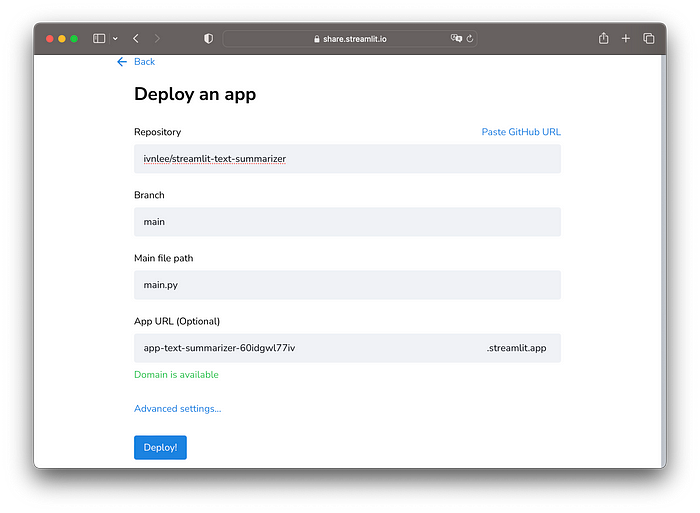
The pipreqs package will automatically create a text file with a list of the libraries required by your app. (Make sure to push all the files to GitHub).

Once all the files are on the GitHub repo, we are ready to deploy. From our Streamlit workspace click on the ***New app*** button and select the option ***From existing repo****:*



Sign in to [share.streamlit.io](https://share.streamlit.io/)

Next, enter the repo URL and the main file path. A URL will be automatically assigned to the app, but this can be changed (depending on availability). Finally, click the ***Deploy!*** button:



And that's it! after a couple of minutes, the app will be deployed and accessible from the web.

There are many other widgets that you can use to add functionality to your apps, I recommend checking the full documentation [here](https://docs.streamlit.io/library/api-reference).

Thanks for reading! I hope you find this article useful, feel free to share it and leave me your comments.