> 使用Streamlit和Hugging Face的免费LLM摘要解锁个人第二大脑：构建在 PC 上运行的 Python Web 应用程序。

作为开发者就是要习惯不停的学习，如果正在阅读本文，那就意味着你和许多开发者一样，想要学习更多，想了解很多有意思的东西是如何运作的，它们是如何连接的。

而在学习的过程中，需要处理大量的信息，阅读大量的书籍：有时候，会不会觉得时间不够用，效率不高？

的确如此，通常需要处理大量的开发文档，需要尽可能多地阅读，以了解这个被称为人工智能的新世界。如果想管理好工作、兴趣和家庭，就需要一个策略。

开始总结材料（文章、电子书籍、博客文章等等），然后在笔记中将它们分类。总结一些列内容的主题，然后快速反馈是否需要深入研究。

在本文中，将展示如何使用HuggingFace和Python使用免费大语言模型（LLM）来构建一个 Summarization 应用程序：使用自己日常没有GPU的电脑。整个过程涉及以下几个步骤:

1. 下载 LaMini 模型
2. 准备 Python 环境并安装依赖
3. 测试汇总 Pipeline
4. 使用 Streamlit 准备并测试图形界面
5. 将逻辑和图形界面整合

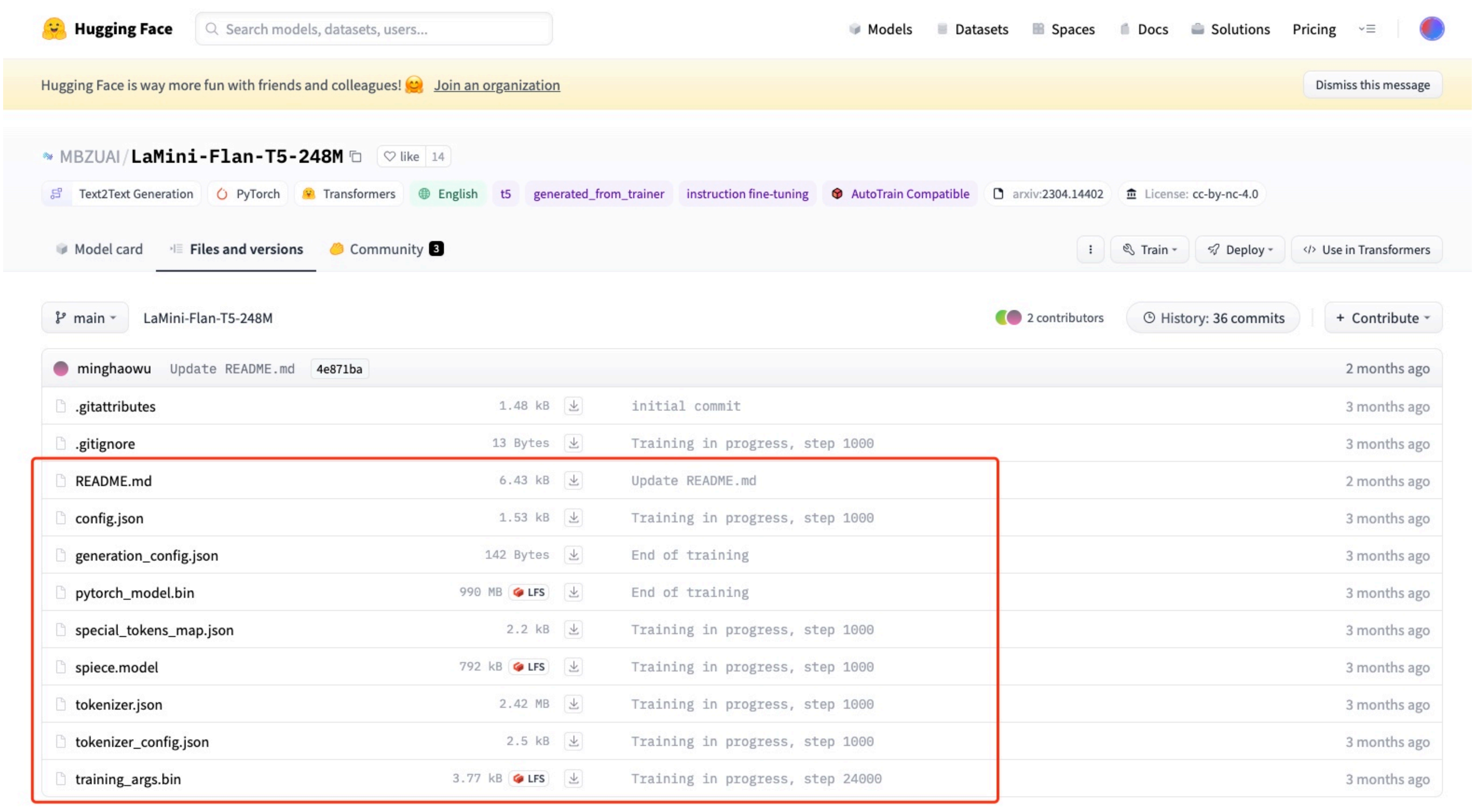
1.下载 LaMini 模型

将使用 Hugging Face 的 `LaMini-LM`：这是基于 `Flan-T5` 系列的小型模型。它拥有 `248M` 参数，在下游 NLP 任务（文本生成、QnA 和摘要）上与 `Aplaca-7B` 和 `LLaMa-7B` 的性能相同。

> 此设置已在运行 Python 3.10 的 `Windows 10 64` 位和 `Mac（Intel 芯片）` 上进行了测试。检查以下链接以获取与其他 Python 版本相关的提示。注意 `torch` 的安装，因为 `pytorch` 是针对特定的 python 版本编译的（下面的链接了解更多信息）

- 为项目创建一个新文件夹 `ai-summarization`
- 在项目目录中创建一个子文件夹 `model`

点击 Hugging Face 存储库 LaMini-Flan-T5-248M，并将目录中的所有文件下载到刚刚创建的 `model` 文件夹中。



minghaowu Update README.md 4e871ba 2 months ago

.gitattributes	1.48 kB	initial commit	3 months ago
.gitignore	13 Bytes	Training in progress, step 1000	3 months ago
README.md	6.43 kB	Update README.md	2 months ago
config.json	1.53 kB	Training in progress, step 1000	3 months ago
generation_config.json	142 Bytes	End of training	3 months ago
pytorch_model.bin	990 MB LFS	End of training	3 months ago
special_tokens_map.json	2.2 kB	Training in progress, step 1000	3 months ago
spiece.model	792 kB LFS	Training in progress, step 1000	3 months ago
tokenizer.json	2.42 MB	Training in progress, step 1000	3 months ago
tokenizer_config.json	2.5 kB	Training in progress, step 1000	3 months ago
training_args.bin	3.77 kB LFS	Training in progress, step 24000	3 months ago

2. 准备Python环境并安装依赖

有很多库需要安装，注意，与 Hugging Face 模型交互的核心是 torch 和 Transformers 库，有关详细信息，请参阅上面提到的链接。

- 在 `ai-summarization` 目录中，创建一个虚拟环境并激活它：

```
python3 -m venv venv
source venv/bin/activate # ubuntu/Mac
venv\Scripts\activate # windows
```

- 在 `venv` 处于激活状态的情况下，安装以下内容：

```
pip install mkl mkl-include # MAC 使用CPU必须安装
pip install torch==1.11.0 torchvision==0.12.0 torchaudio==0.11.0 # 核心
# 安装 Hugging Face Transformer库，需要与LLM进行交互
pip install git+https://github.com/huggingface/transformers

# 这些将在下面用于与文档交互
pip install langchain==0.0.173
pip install faiss-cpu==1.7.4
pip install unstructured==0.6.8
pip install pytesseract==0.3.10
pip install pypdf==3.9.0
pip install pdf2image==1.16.3
pip install sentence_transformers==2.2.2

# 只需要在CPU上运行
pip install accelerate==0.19.0

# 对于GUI和web应用程序
pip install streamlit
```

- 在主目录 `ai-summarization` 中创建一个新 python 文件 `main.py` ，将验证所有库是否已正确安装。

```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
from transformers import pipeline
import torch
import streamlit
```

转到终端，在激活 `venv` 状态下运行 `python3 main.py` 。如果没有看到任何错误，则表示一切都安装成功了。

可能会出现以下异常：

```
ImportError: accelerate>=0.20.3 is required for a normal functioning of this module, but found accelerate==0.19.0.
```

可以重新安装包：

```
pip install accelerate==0.20.3
```

3. 测试摘要管道

进行摘要的方法有很多种：这里将使用管道方法。来自 Transformers 库的 Pipelines 是专用于特定任务（命名实体识别、屏蔽语言建模、情感分析、特征提取和问答）的工具。使用所需的所有导入和管道的初始化更新 `main.py` 文件。

```
########### GUI IMPORTS ################
import streamlit as st
#### IMPORTS FOR AI PIPELINES ###############
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
from transformers import pipeline
from transformers import AutoModel, T5Tokenizer, T5Model
from transformers import T5ForConditionalGeneration
from langchain.llms import HuggingFacePipeline
import torch
```

模型存储在 `checkpoint` （对于本项目来说是目录 `model` ）：它作为编码器-解码器模型，因此将其进行初始化：

```
# 设置 model 路径
```

```
checkpoint = "./model/"  # 实际上是LaMini-Flan-T5-248M
# 初始化标记器和模型
tokenizer = T5Tokenizer.from_pretrained(checkpoint)
base_model = T5ForConditionalGeneration.from_pretrained(
                                    checkpoint,
                                    device_map='auto',
                                    torch_dtype=torch.float32)
```

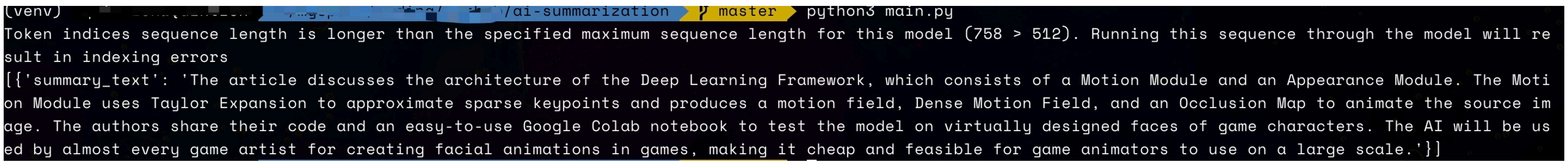
管道指定希望 LLM 执行的任务：设置模型标记器并添加一些特定参数（摘要的 `max_length` 和 `min_length` ）。

```
# 初始化管道
pipe_sum = pipeline('summarization',
                model = base_model,
                tokenizer = tokenizer,
                max_length = 350,
                min_length = 25)
```

为了更好的测试它，将沿着文本分配给一个字符串变量，然后将在其上执行管道：

```
text = "It was way back in 2011 when the game L.A. Noire came out with absolutely amazing life-like facial animations that seemed so ahead of every other game. Now, almost a decade later, we still haven't seen many other games come anywhere close to matching its level in terms of delivering realistic facial expressions.This is because the facial scanning technology used in the development of this game, called MotionScan, was extremely expensive and the file sizes of the captured animations was too big, which is why it made it impractical for most publishers to adopt this technology for their games.Let's take a look at the architecture of this Deep Learning Framework in the figure below. It consists of a Motion Module and an Appearance Module. The driving video is the input to the Motion Module and the Source Image is our target object which is the input to the Appearance Module.The Motion Module consists of an encoder that learns a latent representation containing sparse keypoints of high importance in relation to the motion of the object, which is a face in this scenario. The movement of these keypoints across the different frames of the driving video generate a motion field, which is driven by a function that we want our model to learn. The authors use Taylor Expansion to approximate this function to the first order that creates this motion field. According to the authors, this is the first time first order approximation has been used to model motion. Moreover, learned affine transformations of these keypoints are combined to produce Dense Motion Field. The dense motion field predicts the motion of every individual pixel of the frame, as opposed to focusing on just the keypoints in the sparse motion field. Next, the motion module also produces an Occlusion Map, which highlights the pixels of the frame that need to be in-painted, arising from the movements of the head w.r.t. the background.The Appearance Module uses an encoder to encode the source image, which is then combined with the Motion Field and the Occlusion Map to animate the source image. A Generator model is used for this purpose. During the self-supervised training process, a still frame from the driving video is used as the source image and the learned motion field is used to animate this source image. The actual frames of the video act as the ground truth for the generated motion, hence it is self-supervised training. During the testing/inference phase, this source image can be replaced with any other image from the same object category, and doesn't have to arrive from the driving video.I wanted to explore how well this model works on some virtually designed faces of game characters. The authors have shared its code and an easy-to-use Google Colab notebook to test this out. Here's how their trained model looks when tested on different characters from the game Grand Theft Auto.As you can see, it is extremely easy to create life-like animations with this AI, and I think it will be used by almost every game artist for creating facial animations in games. Moreover, in order to perform Mo-Cap with this technique, all we need now is one camera and any average computer with a GPU and this AI will take care of the rest, making it extremely cheap and feasible for game animators to use this tech on a large scale. This is why I'm excited about the massive improvements that can be brought by this AI in the development of future games."
# 在文本上运行管道并打印结果
result = pipe_sum(text)
print(result)
```

在 `venv` 处于激活状态的情况下，从终端运行 `python3 main.py`，将会看到下面这样的结果。



```
(venv) [redacted]/ai-summarization  master  python3 main.py
Token indices sequence length is longer than the specified maximum sequence length for this model (758 > 512). Running this sequence through the model will re
sult in indexing errors
[{'summary_text': 'The article discusses the architecture of the Deep Learning Framework, which consists of a Motion Module and an Appearance Module. The Moti
on Module uses Taylor Expansion to approximate sparse keypoints and produces a motion field, Dense Motion Field, and an Occlusion Map to animate the source im
age. The authors share their code and an easy-to-use Google Colab notebook to test the model on virtually designed faces of game characters. The AI will be us
ed by almost every game artist for creating facial animations in games, making it cheap and feasible for game animators to use on a large scale.'}]
```

由于是一个小的测试，存在一些错误：

```
Token indices sequence length is longer than the specified maximum sequence length for this model (758 > 512). Running this sequence through the model will result in indexing errors
```

这是因为原文长度有点长，稍后会对其进行切割。请注意，管道的结果是一个带有字典的列表：因此，要仅调用文本字符串，应该使用 `[0]` 作为列表中的第一个项目，而 `['summary_text']` 是想要的值（字符串）的键。

```
print(result[0]['summary_text'])
```

4. 使用 Streamlit 准备并测试图形界面

现在逻辑部分已经完成（除了文本分割器），接下来将深入研究 Streamlit 应用程序。

Streamlit 是一个用于构建数据 Web 应用程序的库，无需了解任何前端技术（例如 HTML 和 CSS）。如果想了解更多信息，请点击此处查看文档。

创建一个名为 `TextSummarizer-webui.py` 的 python 文件：首先创建 GUI 的主干，然后将元素与逻辑结合起来。

```
import streamlit as st
############# Displaying images on the front end #################
st.set_page_config(page_title="Mockup for single page webapp",
               page_icon='💻',
               layout="centered",  #or wide
               initial_sidebar_state="expanded",
               menu_items={
                   'Get Help': 'https://docs.streamlit.io/library/api-reference',
                   'Report a bug': "https://www.extremelycoolapp.com/bug",
                   'About': "# This is a header. This is an *extremely* cool app!"}
                   )
# Load image placeholder from the web
st.image('./images/header.png', width=750)
# Set a Descriptive Title
st.title("My AI Summarizer")
st.divider()
your_future_text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras rhoncus massa sit amet est congue dapibus. Duis dictum ac nulla sit amet sollicitudin. In non metus ac neque vehicula egestas. Vestibulum quis justo id enim vestibulum venenatis. Cras gravida ex vitae dignissim suscipit. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis efficitur, lorem ut fringilla commodo, lacus orci lobortis turpis, sit amet consequat ante diam ut libero."
st.text_area('Summarized text', your_future_text,
          height = 150, key = 'result')
```

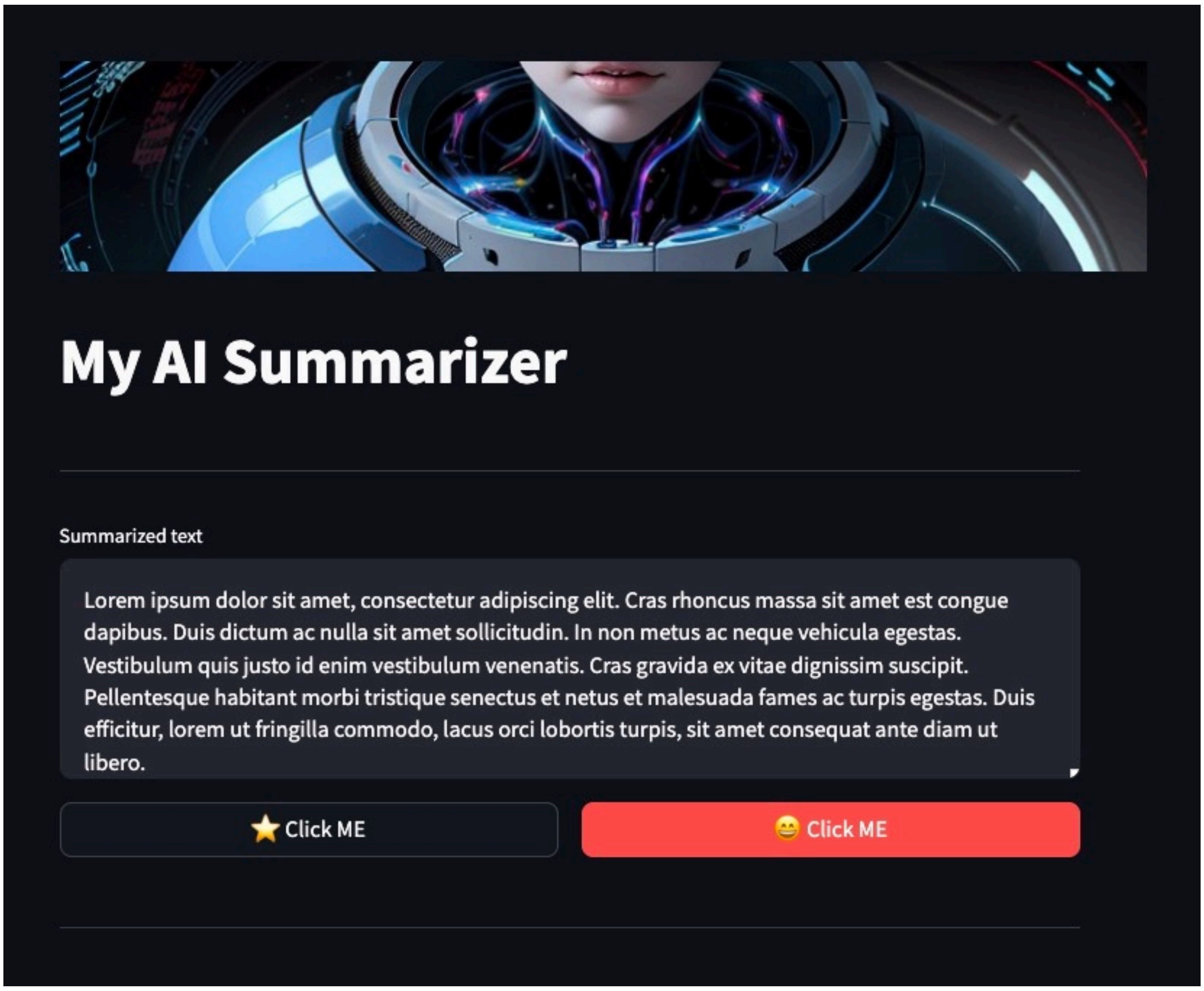
- 导入 streamlit 后，第一条语句必须是 set_page_config（如果将其放在程序中的其他位置，则会抛出错误）：参数是 `webapp` 页面的总体布局的设置。
- 然后设置 header image：仅用于测试，在这里使用来自网络的图像 `https://placehold.co/750x150`。
- st.text_area是另一个 Stramlit 小部件：它创建一个带有标题和内容的文本区域。在这里的例子中，内容将由字符串 `your_future_text` 中的文本填充。
- 最后一个参数是 `key = 'result'`：将用它来调用 `session_states` （应用程序运行时可以调用和更新变量的方式）

```
# Set 2 colums to make the Buttons wider
col1, col2 = st.columns(2)
btn1 = col1.button(" :star: Click ME ", use_container_width=True, type="secondary")
btn2 = col2.button(" :smile: Click ME ", use_container_width=True, type="primary")

if btn1:
   st.warning('You pressed the wrong one!', icon="⚠️")
if btn2:
   st.success('Good Choice!', icon="⚠️")
st.divider()
```

- 对于本示例，仅在此定义 2 列，并在每一列中放置一个按钮。当在容器（列）内时，调用的小部件不带 `st.`。使用 `use_container_width=True` 将 Button 的宽度扩展到列之一。
- 保存所有内容，在终端并运行 `Streamlit` 应用程序类型：`streamlit run TextSummarizer-webui.py`

默认浏览器将在默认地址 `http://localhost:8501` 打开。



My AI Summarizer

Summarized text

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras rhoncus massa sit amet est congue dapibus. Duis dictum ac nulla sit amet sollicitudin. In non metus ac neque vehicula egestas. Vestibulum quis justo id enim vestibulum venenatis. Cras gravida ex vitae dignissim suscipit. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Duis efficitur, lorem ut fringilla commodo, lacus orci lobortis turpis, sit amet consequat ante diam ut libero.

5.将逻辑和界面联调起来

简单介绍完 Streamlit 之后，再来说逻辑部分（AI pipeline）和图形用户界面部分（Streamlit）。不用担心代码：可以在 GitHub 存储库中找到它。

重命名之前的文件 `main.py` 并创建一个新文件 `AI-TextSummarizer.py`，代码如下：

```
########### GUI IMPORTS ################
import streamlit as st
import ssl
############# Displaying images on the front end #################
st.set_page_config(page_title="Summarize and Talk ot your Text",
              page_icon='📖',
              layout="centered",  #or wide
              initial_sidebar_state="expanded",
              menu_items={
                  'Get Help': 'https://docs.streamlit.io/library/api-reference',
                  'Report a bug': "https://www.extremelycoolapp.com/bug",
                  'About': "# This is a header. This is an AI text summarizer!"
                      },
              )
########### SSL FOR PROXY ##############
ssl._create_default_https_context = ssl._create_unverified_context

#### IMPORTS FOR AI PIPELINES ###############
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
from transformers import pipeline

from transformers import AutoModel, T5Tokenizer, T5Model
from transformers import T5ForConditionalGeneration
from langchain.llms import HuggingFacePipeline
import torch
import datetime

#############################################################################
```

```
#             SIMPLE TEXT2TEXT GENERATION INFERENCE
#           checkpoint = "./models/LaMini-Flan-T5-783M.bin"
# ###########################################################################
checkpoint = "./model/"  # 实际上是 LaMini-Flan-T5-248M
```

到目前为止没有什么新的。将在以下代码块中将函数和交互式 Streamlit widt 放在一起，并解释构建块。

```
########################################################################
#     SUMMARIZATION FROM TEXT STRING WITH HUGGINGFACE PIPELINE       #
########################################################################
def AI_SummaryPL(checkpoint, text, chunks, overlap):

    """
    checkpoint is in the format of relative path
    example:  checkpoint = "/content/model/"  #it is actually LaMini-Flan-T5-248M   #tested fine
    text it is either a long string or a input long string or a loaded document into string
    chunks: integer, lenght of the chunks splitting
    ovelap: integer, overlap for cor attention and focus retreival
    RETURNS full_summary (str), delta(str) and reduction(str)

    post_summary14 = AI_SummaryPL(LaMini,doc2,3700,500)
    USAGE EXAMPLE:
    post_summary, post_time, post_percentage = AI_SummaryPL(LaMini,originalText,3700,500)
    """
    from langchain.text_splitter import RecursiveCharacterTextSplitter
    text_splitter = RecursiveCharacterTextSplitter(
        # 设置一个非常小的块大小，只是为了显示
        chunk_size = chunks,
        chunk_overlap  = overlap,
        length_function = len,
    )
    texts = text_splitter.split_text(text)
    checkpoint = checkpoint
    tokenizer = T5Tokenizer.from_pretrained(checkpoint)
    base_model = T5ForConditionalGeneration.from_pretrained(checkpoint,
                                                device_map='auto',
                                                torch_dtype=torch.float32)
    ### INITIALIZING PIPELINE
    pipe_sum = pipeline('summarization',
                        model = base_model,
                        tokenizer = tokenizer,
                        max_length = 350,
                        min_length = 25
                        )
    ## START TIMER
    start = datetime.datetime.now()
    ## START CHUNKING
    full_summary = ''
    for cnk in range(len(texts)):
      result = pipe_sum(texts[cnk])
      full_summary = full_summary + ' '+ result[0]['summary_text']
    stop = datetime.datetime.now()
    ## TIMER STOPPED AND RETURN DURATION
    delta = stop-start
    ### Calculating Summarization PERCENTAGE
    reduction = '{:.1%}'.format(len(full_summary)/len(text))
    print(f"Completed in {delta}")
    print(f"Reduction percentage: ", reduction)

    return full_summary, delta, reduction
```

这是主要的功能，接下来需要一个函数，因为单击正确的按钮时将开始汇总（对于此方法，需要一个函数来调用）

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
text_splitter = RecursiveCharacterTextSplitter(
    # 设置一个非常小的块大小，只是为了显示
    chunk_size = chunks,
    chunk_overlap  = overlap,
    length_function = len,
)
```

```
texts = text_splitter.split_text(text)
```

LangChain 库是一个非常强大的工具箱：可以使用外部文档和来源与语言模型进行交互。LangChain TextSplitters 的方法不止一种。RecursiveCharacterSplitter 是推荐的一种，用于将长通用文本分割成小块（称为块），并且不超过 token 限制。

```
## 开始组块
full_summary = ''
for cnk in range(len(texts)):
  result = pipe_sum(texts[cnk])
  full_summary = full_summary + ' '+ result[0]['summary_text']
```

块存储在列表中：迭代列表中的项目并将每个块提供给摘要管道。然后将所有字符串连接在一起以获得 `final_summary` 。

```
### HEADER section
st.image('./images/header.png', width=750)
st.title("My AI Summarizer")
st.divider()
title = st.text_area('Insert here your Copy/Paste text', "", height = 350, key = 'copypaste')
btt = st.button("1. Start Summarization")
txt = st.empty()
timedelta = st.empty()
text_lenght = st.empty()
redux_bar = st.empty()
st.divider()
down_title = st.empty()
down_btn = st.button('2. Download Summarization')
text_summary = ''
```

可以看到一些 `st.empty()` 。这是一个占位符：正在页面布局中 `预订` 一个位置，稍后将填充该位置。

```
def start_sum(text):
    if st.session_state.copypaste == "":
        st.warning('You need to paste some text...', icon="⚠️")
    else:
        with st.spinner('Initializing pipelines...'):
            st.success(' AI process started', icon="🤖")
            print("Starting AI pipelines")
            text_summary, duration, reduction = AI_SummaryPL(LaMini,text,3700,500)
        txt.text_area('Summarized text', text_summary, height = 350, key='final')
        timedelta.write(f'Completed in {duration}')
        text_lenght.markdown(f"Initial length = {len(text.split(' '))} words / summarization = **{len(text_summary.split(' '))} words**")
        redux_bar.progress(len(text_summary)/len(text), f'Reduction: **{reduction}**')
        down_title.markdown(f"## Download your text Summarization")
```

当按下 `btt = st.button("1. Start Summarization")` 时将调用此函数，开始对粘贴在 `text_area` 中的文本进行摘要。

```
if btt:
    start_sum(st.session_state.copypaste)

if down_btn:
    def savefile(generated_summary, filename):
        st.write("Download in progress...")
        with open(filename, 'w') as t:
            t.write(generated_summary)
        t.close()
        st.success(f'AI Summarization saved in {filename}', icon="✅")
    savefile(st.session_state.final, 'text_summarization.txt')
    txt.text_area('Summarized text', st.session_state.final, height = 350)
```

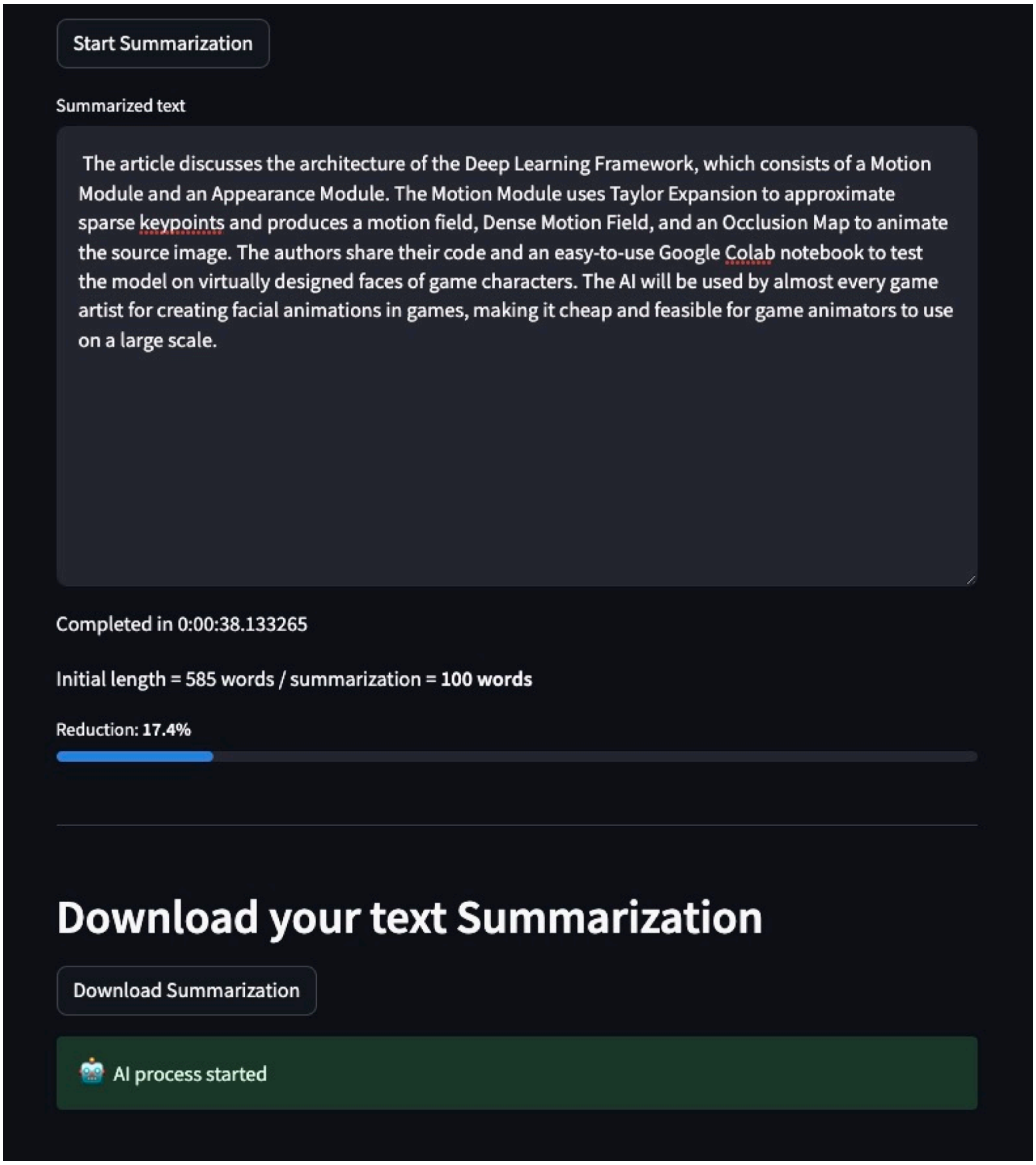
> 请注意， `start_sum` 的唯一参数是 `session_state` 。

> Session State 是一种在每个用户会话的重新运行之间共享变量的方法。除了存储和持久状态的能力之外，Streamlit 还公开了使用回调操作状态的能力。会话状态也会在多页面应用程序内的应用程序之间持续存在。

在 `venv` 处于活动状态时，从终端运行：

```
streamlit run AI-TextSummarizer.py
```

粘贴想要总结的文章文本，然后按按钮。



The article discusses the architecture of the Deep Learning Framework, which consists of a Motion Module and an Appearance Module. The Motion Module uses Taylor Expansion to approximate sparse keypoints and produces a motion field, Dense Motion Field, and an Occlusion Map to animate the source image. The authors share their code and an easy-to-use Google Colab notebook to test the model on virtually designed faces of game characters. The AI will be used by almost every game artist for creating facial animations in games, making it cheap and feasible for game animators to use on a large scale.

Completed in 0:00:38.133265

Initial length = 585 words / summarization = **100 words**

Reduction: **17.4%**

Download your text Summarization

AI process started

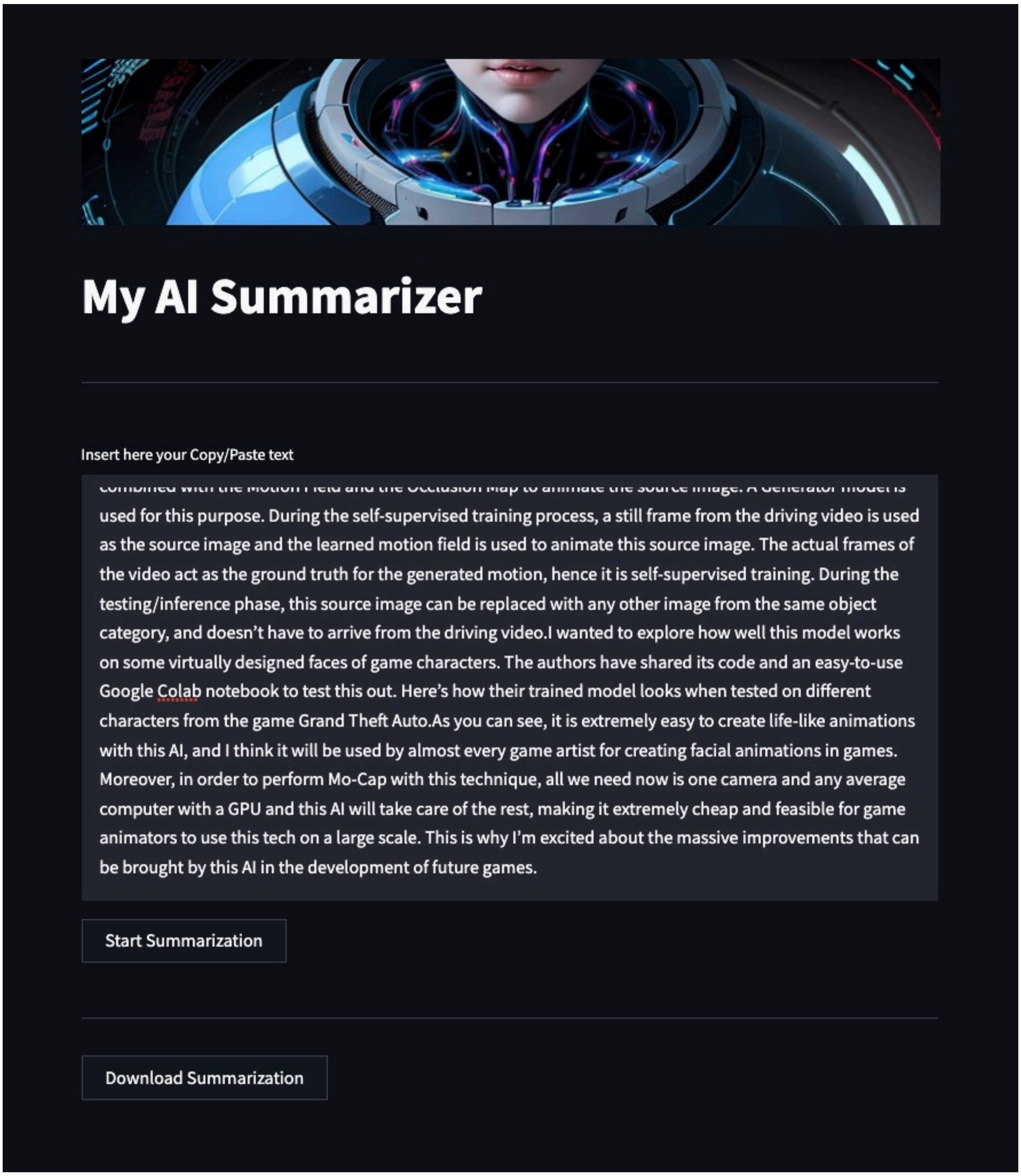
下面尝试增加一些自定义样式：

```
custom_style = """
    <style>
        .stButton>button {
            border-radius:0px;
            padding:5px 20px;
        }
        .stTextArea textarea {
            border-radius:0px;
            padding:5px;
            line-height:1.6
        }
        .stTextArea .st-br{
            border-radius:0px;
        }
        .stTextArea>label p{
            line-height:2
        }
        .block-container{
            max-width:780px
        }
    </style>
"""
```

```

st.markdown(custom_style, unsafe_allow_html=True)
```

效果如下：



My AI Summarizer

Insert here your Copy/Paste text

combined with the Motion Field and the Occlusion Map to animate the source image. A Generator model is used for this purpose. During the self-supervised training process, a still frame from the driving video is used as the source image and the learned motion field is used to animate this source image. The actual frames of the video act as the ground truth for the generated motion, hence it is self-supervised training. During the testing/inference phase, this source image can be replaced with any other image from the same object category, and doesn't have to arrive from the driving video.I wanted to explore how well this model works on some virtually designed faces of game characters. The authors have shared its code and an easy-to-use Google Colab notebook to test this out. Here's how their trained model looks when tested on different characters from the game Grand Theft Auto.As you can see, it is extremely easy to create life-like animations with this AI, and I think it will be used by almost every game artist for creating facial animations in games. Moreover, in order to perform Mo-Cap with this technique, all we need now is one camera and any average computer with a GPU and this AI will take care of the rest, making it extremely cheap and feasible for game animators to use this tech on a large scale. This is why I'm excited about the massive improvements that can be brought by this AI in the development of future games.

总结

管道 Pipelines 是惊人的。即使硬件很少，也可以在计算机上运行我们想要的所有内容（LaMini-LM 也仅使用 CPU 运行）。尝试不同的设置以提高摘要的质量。