```
##### `🤖_OpenAI.py`
##### OpenAI ChatGPT Demo
##### Open-Source, hosted on https://github.com/DrBenjamin/OpenAI
##### Please reach out to ben@benbox.org for any questions
#### Loading needed Python libraries
import streamlit as st
import streamlit_scrollable_textbox as sty
import io
import openai
import PyPDF2
from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont




#### Streamlit initial setup
st.set_page_config(
   page_title = "OpenAI",
   page_icon = "https://www.benbox.org/R/OpenAI.png",
   layout = "centered",
   initial_sidebar_state = "expanded"
)




#### Session states
if 'benbox' not in st.session_state:
   st.session_state['benbox'] = ''
if 'messages' not in st.session_state:
   st.session_state['messages'] = ''
if 'system' not in st.session_state:
   st.session_state['system'] = ''
if 'temp' not in st.session_state:
   st.session_state['temp'] = .3
if 'token' not in st.session_state:
   st.session_state['token'] = 128
if 'used_tokens' not in st.session_state:
   st.session_state['used_tokens'] = 0
if 'demo' not in st.session_state:
   st.session_state['demo'] = False
if 'chat' not in st.session_state:
   st.session_state['chat'] = 0
```

```
#### Functions
### Function benbox() = Write costs to image
def benbox(image, text, text2, text3):
    # Open an Image
    img = Image.open(image)

    # Call draw Method to add 2D graphics in an image
    I1 = ImageDraw.Draw(img)
    I1.text((12, 16), text, font = ImageFont.truetype('images/Menlo.ttc', 14), fill = (158, 38, 26))
    I1.text((12, 48), text2, font = ImageFont.truetype('images/Menlo.ttc', 14), fill = (158, 38,
26))
    I1.text((12, 80), text3, font = ImageFont.truetype('images/Menlo.ttc', 14), fill = (158, 38,
26))

    # Save the edited image to buffer
    s = io.BytesIO()
    img.save(s, 'png')
    st.session_state['benbox'] = s.getvalue()




#### Main program
st.header('OpenAI ChatGPT language model')

# Set API key
openai.api_key = st.secrets['openai']['key']
answer = ''
used_tokens = 0


### PDF documents
st.subheader('Use data from a PDF source')
pdf_text = ' """'
index = 0
pdf_usage = st.checkbox('Include a PDF source to feed ChatGPT with data?')
if pdf_usage:
    st.write('**:green[Use your own PDF]**')
    documents = ["ChatGPT.pdf", "LEAM.pdf", "LLM.pdf", "KW.pdf", "AIDH.pdf", "PC.pdf"]
    uploaded_file = st.file_uploader(label = 'Choose a PDF file to upload', type = 'pdf')
    if uploaded_file is not None:
        file_name = os.path.join('PDFs', uploaded_file.name)
        file = open(file_name, 'wb')
        file.write(uploaded_file.getvalue())
        file.close()
        documents.append(uploaded_file.name)
        index = len(documents) - 1
    ## Source selection
    st.write('**:green[or a provided PDF about AI / Programming]**')
    pdf = st.selectbox(label = 'Choose PDF document?', options = documents, index = index)

    # Creating a pdf reader object
    reader = PyPDF2.PdfReader('PDFs/' + pdf)

    # Select pages
    if len(reader.pages) > 1:
        pages_range = st.slider(label = 'Select a range of pages you want to use for the dialog
with ChatGPT',
                        min_value = 1, max_value = len(reader.pages), value = (1, 1))

        # print the text of the first page
        pagez = []
        for i in range(pages_range[0], pages_range[1], 1):
            pagez.append(i)
            pdf_text += reader.pages[i].extract_text()
        pdf_text += '"""'
        page = st.radio(
            label = 'Page preview for selecting meaningful pages of the PDF source (use slider
above to adjust)',
            options = pagez, horizontal = True, index = 0)
        if page is None:
            page = 0
        sty.scrollableTextbox(reader.pages[page].extract_text(), height = 256, border = True)
    else:
        pdf_text += reader.pages[0].extract_text() + '"""'
        sty.scrollableTextbox(reader.pages[0].extract_text(), height = 256, border = True)



### OpenAI ChatGPT
## Model selection
st.subheader('Choose a model')
model = st.selectbox(label = 'What model to use?', options = ["gpt-4-0314", "gpt-3.5-turbo",
"text-davinci-003", "text-curie-001", "text-babbage-001", "text-ada-001", "whisper-1"], index
= 0)

# Show info about model and set variable costs
chat_usage = False
if model == "gpt-4-0314":
    cost_co_eff = 0.05
    st.write(':green[Capability of this model:] More capable than any GPT-3.5 model, able to
do more complex tasks, and optimized for chat. Will be updated with our latest model
iteration. :red[Costs:] \$0.03/1k for prompt tokens and \$0.06/1k for sampled tokens.')
    chat_usage = st.checkbox('Have an ongoing Chat?')
```

```
elif model == "gpt-3.5-turbo":
    cost_co_eff = 0.002
    st.write(':green[Capability of this model:] Best model, also for many non-chat use cases.
:red[Costs:] ' + str(cost_co_eff) + '$ (per 1K tokens)')
    chat_usage = st.checkbox('Have an ongoing Chat?')
elif model == "text-davinci-003":
    cost_co_eff = 0.02
    st.write(
        ':green[Capability of this model:] Most capable GPT-3 model. Can do any task the other
models can do, often with higher quality, longer output and better instruction-following.
Also supports inserting completions within text. :red[Costs:] ' + str(
            cost_co_eff) + '$ (per 1K tokens)')
elif model == "text-curie-001":
    cost_co_eff = 0.002
    st.write(':green[Capability of this model:] Very capable, but faster and lower cost than
Davinci. :red[Costs:] ' + str(
        cost_co_eff) + '$ (per 1K tokens)')
elif model == "text-babbage-001":
    cost_co_eff = 0.0005
    st.write(':green[Capability of this model:] Capable of straightforward tasks, very fast, and
lower cost. :red[Costs:] ' + str(
        cost_co_eff) + '$ (per 1K tokens)')
elif model == "text-ada-001":
    cost_co_eff = 0.0004
    st.write(
        ':green[Capability of this model:] Capable of very simple tasks, usually the fastest model
in the GPT-3 series, and lowest cost. :red[Costs:] ' + str(
            cost_co_eff) + '$ (per 1K tokens)')
elif model == "whisper-1":
    cost_co_eff = 0.006
    st.write(':green[Capability of this model:] Whisper is an automatic speech recognition
(ASR) system trained on 680,000 hours of multilingual and multitask supervised data
collected from the web. :red[Costs:] ' + str(cost_co_eff) + '$ (per 1K tokens)')
else:
    cost_co_eff = 0.02


## Show Configuration and Explanations
if not chat_usage or st.session_state['chat'] < 2:
    if model == "whisper-1":
        with st.form('Whisperer'):
            st.subheader('Audio 2 Text')
            audio_file = st.file_uploader(label = "Upload an audio file", type = 'mp3')
            submitted = st.form_submit_button('Submit')
            if submitted:
                # Check for audio data
                if audio_file is not None:
```

```
                st.subheader('Preview used audio')
                st.audio(audio_file)
                transcript = openai.Audio.transcribe(model, audio_file)
                st.markdown('Transcript: :orange[' + str(transcript['text']) + ']')
    else:
        # Columns
        col1, col2 = st.columns(2)
        with col1:
            ## Form (to prevent unnecessary requests)
            with st.form("OpenAI"):
                # Text input
                st.subheader('Communicate')
                question = st.text_input('What question do you want to ask OpenAI ChatGPT?')

                # Temperature selection
                temp = st.slider('Which temperature?', 0.0, 1.0, .3)

                # Tokens selection
                tokens = st.slider("Answer's max tokens", 1, 4000, 128)

                # Store if Chat-Bot
                if chat_usage:
                    st.session_state['temp'] = temp
                    st.session_state['token'] = tokens
                    st.session_state['chat'] += 1



                ## Submit button
                submitted = st.form_submit_button('Submit')
                if submitted:
                    try:
                        if not chat_usage:
                            # Using ChatGPT from OpenAI
                            if model == 'gpt-3.5-turbo' or model == 'gpt-4-0314':
                                response_answer = openai.ChatCompletion.create(
                                    model = model,
                                    messages = [
                                        {"role": "system", "content": "You are a helpful assistant."},
                                        {"role": "user", "content": question + pdf_text},
                                    ]
                                )
                                answer = response_answer['choices'][0]['message']['content']
                            else:
                                response_answer = openai.Completion.create(model = model, prompt =
question + pdf_text, temperature = temp,
                                                        max_tokens = tokens, top_p = 1.0,
frequency_penalty = 0.0,
                                                presence_penalty = 0.0, stop = ["\"\"\""])
                     answer = response_answer['choices'][0]['text']
                   used_tokens = response_answer['usage']['total_tokens']
                 else:
                   st.session_state['system'] = question
                   st.experimental_rerun()
               except Exception as e:
                 if str(e) == 'That model does not exist':
                   st.error(body = str(e) + ' in the open API, wait for the beta access!', icon = 
"🚨")
                 else:
                   st.error(body = e, icon = "🚨")
        with col2:
          st.subheader('Examples')
          if pdf_usage:
            st.markdown('If you included PDF data type in something like\n\n*:orange[Please 
summarise this:]*\n\nor\n\n*:orange[Summarise this in 5 sentences:]*')
          else:
            if model == "text-curie-001":
              st.markdown(
                'If choosen "Curie" model you can type in something like\n\n*:orange[Extract a 
keyword in this text "Saturdays it is often raining!"]*')
            elif model == "text-babbage-001":
              st.markdown(
                'If choosen "Babbage" model you can type in something 
like\n\n*:orange[Improve this text "Saturdays it is often raining!"]*')
            elif model == "text-ada-001":
              st.markdown(
                'If choosen "Ada" model you can type in something like\n\n*:orange[Rephrase 
this text "Saturdays it is often raining!"]*')
            else:
              if chat_usage:
                st.markdown('Type in something like\n\n*:orange[You are a helpful assistant] 
or :orange[You are a cynical and humorous assistant.]*')
                st.write('or use the Demo')
                st.session_state['demo'] = st.checkbox(label = 'BenBox Demo (press submit)')
              else:
                st.markdown('Type in something like\n\n*:orange[Write me a short poem] or 
:orange[Write a Python program about fibonacci]*')
            st.markdown('**Temperature**\n\n:green[*0 = each answer will be the 
same*]\n\n:green[*1 = more "creative" answers*]')
            if not chat_usage:
              st.markdown('**Tokens**\n\n:green[*1 token ~= 4 chars in English*]')


  ### Outside columns
```

```
    ## Show answer
    if answer != '':
        sty.scrollableTextbox(answer, height = 128, border = True)

        # Show costs per query
        costs = float(used_tokens) / 1000 * cost_co_eff
        if costs >= 0.0001:
            st.write(':red[Costs: ' + str(round(costs, 4)) + '$]')
        else:
            st.write(':red[Costs: < 0.0001$]')


## Chat-Bot
else:
    if st.session_state['messages'] == '':
        if st.session_state['demo']:
            st.session_state['temp'] = 0.9
            st.session_state['token'] = 300
            messages_input = [
                {"role": "system", "content": 'You are Ben, an Artificial Intelligence. You are in a blue
Box ("BenBox") and live together with a spaceship crew ("Emmie", "Sertan" and the two
Robots "SAM" and "SEB") on a starship called "Pulp". One of the robots, SAM, is talking to
you right now. You are funny and a quite nice guy and gives short but useful answers. Right
now the Pulp is landed on Earth, in Germany in the city Cologne. The next mission is to find
artifacts in the old big roman churches there.'},
                {"role": "assistant", "content": "What's up?"},
                {"role": "user", "content": "Hi I am SAM, one of the robots on board of the
spaceship Pulp. I am a quite young robot and need to learn a lot of things."},
                {"role": "assistant", "content": "That's cool, I can send a around 100 Trillian
Gigabytes of language model data. Your language processor definitely needs an update!"}
            ]
        else:
            messages_input = [
                {"role": "system", "content": st.session_state['system']},
                {"role": "assistant", "content": "How can I help you?"}
            ]
    else:
        messages_input = st.session_state['messages']
    with st.form('BenBox'):
        if st.session_state['demo']:
            benbox(image = st.secrets['benbox']['image'], text = 'Costs of this Chat with Ben are
reasonable ' + str(round(st.session_state['used_tokens'] / 1000 * 0.002, 4)) + '$', text2 =
'Please pay NOT directly to the Chat-Bot!!!', text3 = 'He always buys `Chips` from the
money...')
            if st.session_state['benbox'] != '':
                st.image(st.session_state['benbox'])
            else:
```

```
        st.image(st.secrets['benbox']['image'])
    for i in range(len(messages_input)):
      if i > 0:
        if i % 2 == 1:
          st.write(':blue[BenBox:] ', messages_input[i]['content'])
        elif i % 2 == 0:
          if st.session_state['demo']:
            st.write(':green[SAM:] ', messages_input[i]['content'])
          else:
            st.write(':green[User:] ', messages_input[i]['content'])
    user_input = st.text_input(label = '', label_visibility = 'collapsed')
    if user_input == 'Exit' or user_input == 'exit' or user_input == 'Quit' or user_input ==
'quit':
      st.session_state['chat'] = 0
      st.session_state['messages'] = ''
      st.experimental_rerun()
    if not st.session_state['demo']:
      st.markdown(':orange[If you are tired talking to the Chat-Bot just type "Quit" or
"Exit".]')
      st.markdown(':red[Costs of this Chat are ' + str(round(st.session_state['used_tokens']
/ 1000 * 0.002, 4)) + '$]')
    else:
      st.markdown(':orange[If you are tired talking to Ben just type "Quit" or "Exit" (he will
be in a huff).]')


    ## Submit button
    submitted = st.form_submit_button('Submit')
    if submitted:
      messages_input.append({"role": "user", "content": user_input})
      try:
        response_answer = openai.ChatCompletion.create(model = model, messages =
messages_input, temperature = st.session_state['temp'], max_tokens =
st.session_state['token'])
        answer = response_answer['choices'][0]['message']['content']
        st.session_state['used_tokens'] += response_answer['usage']['total_tokens']
        messages_input.append({"role": "assistant", "content": answer})
        st.session_state['messages'] = messages_input
        st.experimental_rerun()
      except Exception as e:
        if str(e) == 'That model does not exist':
          st.error(body = str(e) + ' in the open API, wait for the beta access!', icon = "🚨")
        else:
          st.error(body = e, icon = "🚨")
```