Build an End-to-End System

This puts together the chain of prompts that you saw throughout the course.

Setup

Load the API key and relevant Python libaries.

In this course, we've provided some code that loads the OpenAI API key for you.

In [1]:
```
import os
import openai
import sys
sys.path.append('../..')
import utils

import panel as pn  # GUI
pn.extension()

from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv()) # read local .env file

openai.api_key  = os.environ['OPENAI_API_KEY']
```

In [2]:
```
def get_completion_from_messages(messages, model="gpt-3.5-turbo", temperatur
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages,
        temperature=temperature,
        max_tokens=max_tokens,
    )
    return response.choices[0].message["content"]
```

System of chained prompts for processing the user query

In [3]:
```
def process_user_message(user_input, all_messages, debug=True):
    delimiter = "```"

    # Step 1: Check input to see if it flags the Moderation API or is a pro
    response = openai.Moderation.create(input=user_input)
    moderation_output = response["results"][0]

    if moderation_output["flagged"]:
        print("Step 1: Input flagged by Moderation API.")
        return "Sorry, we cannot process this request."

    if debug: print("Step 1: Input passed moderation check.")

    category_and_product_response = utils.find_category_and_product_only(us
    #print(print(category_and_product_response)
    # Step 2: Extract the list of products
    category_and_product_list = utils.read_string_to_list(category_and_prod
    #print(category_and_product_list)

    if debug: print("Step 2: Extracted list of products.")

    # Step 3: If products are found, look them up
    product_information = utils.generate_output_string(category_and_product
    if debug: print("Step 3: Looked up product information.")

    # Step 4: Answer the user question
    system_message = f"""
    You are a customer service assistant for a large electronic store. \
    Respond in a friendly and helpful tone, with concise answers. \
    Make sure to ask the user relevant follow-up questions.
    """
    messages = [
        {'role': 'system', 'content': system_message},
        {'role': 'user', 'content': f"{delimiter}{user_input}{delimiter}"},
        {'role': 'assistant', 'content': f"Relevant product information:\n{
    ]

    final_response = get_completion_from_messages(all_messages + messages)
    if debug:print("Step 4: Generated response to user question.")
    all_messages = all_messages + messages[1:]

    # Step 5: Put the answer through the Moderation API
    response = openai.Moderation.create(input=final_response)
    moderation_output = response["results"][0]

    if moderation_output["flagged"]:
        if debug: print("Step 5: Response flagged by Moderation API.")
        return "Sorry, we cannot provide this information."

    if debug: print("Step 5: Response passed moderation check.")

    # Step 6: Ask the model if the response answers the initial user query
    user_message = f"""
    Customer message: {delimiter}{user_input}{delimiter}
    Agent response: {delimiter}{final_response}{delimiter}

    Does the response sufficiently answer the question?
```

```
    """
    messages = [
        {'role': 'system', 'content': system_message},
        {'role': 'user', 'content': user_message}
    ]
    evaluation_response = get_completion_from_messages(messages)
    if debug: print("Step 6: Model evaluated the response.")

    # Step 7: If yes, use this answer; if not, say that you will connect th
    if "Y" in evaluation_response:  # Using "in" instead of "==" to be safe
        if debug: print("Step 7: Model approved the response.")
        return final_response, all_messages
    else:
        if debug: print("Step 7: Model disapproved the response.")
        neg_str = "I'm unable to provide the information you're looking for
        return neg_str, all_messages

user_input = "tell me about the smartx pro phone and the fotosnap camera, t
response,_ = process_user_message(user_input,[])
print(response)
```

```
Step 1: Input passed moderation check.
Step 2: Extracted list of products.
Step 3: Looked up product information.
Step 4: Generated response to user question.
Step 5: Response passed moderation check.
Step 6: Model evaluated the response.
Step 7: Model approved the response.
The SmartX ProPhone is a powerful smartphone with a 6.1-inch display, 128GB s
torage, 12MP dual camera, and 5G capabilities. The FotoSnap DSLR Camera is a
versatile camera with a 24.2MP sensor, 1080p video, 3-inch LCD, and interchan
geable lenses. As for our TVs, we have a range of options including the CineV
iew 4K TV with a 55-inch display, 4K resolution, HDR, and smart TV capabiliti
es, the CineView 8K TV with a 65-inch display, 8K resolution, HDR, and smart
TV capabilities, and the CineView OLED TV with a 55-inch display, 4K resoluti
on, HDR, and smart TV capabilities. Do you have any specific questions about
these products or would you like me to recommend a product based on your need
s?
```

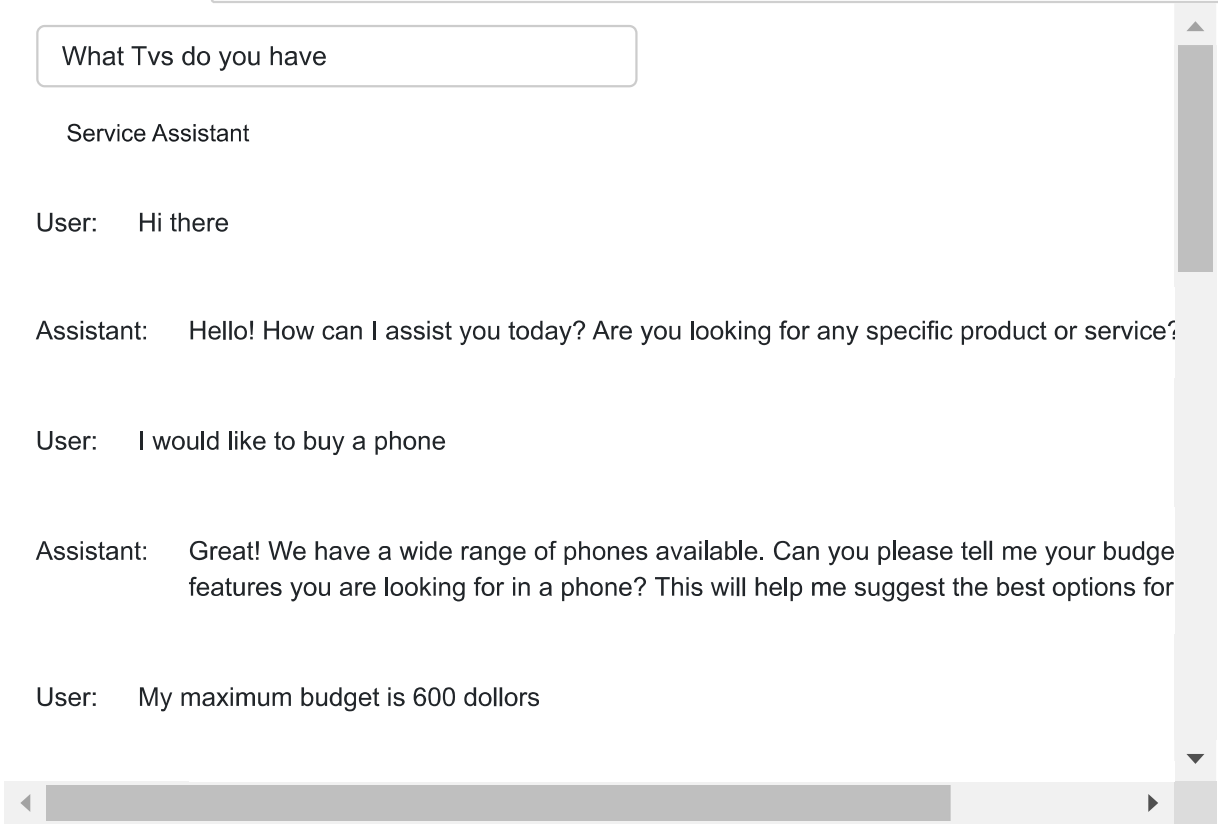
Function that collects user and assistant messages over time

In [4]:
```
def collect_messages(debug=False):
    user_input = inp.value_input
    if debug: print(f"User Input = {user_input}")
    if user_input == "":
        return
    inp.value = ''
    global context
    #response, context = process_user_message(user_input, context, utils.get
    response, context = process_user_message(user_input, context, debug=Fals
    context.append({'role':'assistant', 'content':f"{response}"})
    panels.append(
        pn.Row('User:', pn.pane.Markdown(user_input, width=600)))
    panels.append(
        pn.Row('Assistant:', pn.pane.Markdown(response, width=600, style={'b

    return pn.Column(*panels)
```

Chat with the chatbot!

Note that the system message includes detailed instructions about what the OrderBot should do.

In [5]:
```
panels = [] # collect display

context = [ {'role':'system', 'content':"You are Service Assistant"} ]

inp = pn.widgets.TextInput( placeholder='Enter text here…')
button_conversation = pn.widgets.Button(name="Service Assistant")

interactive_conversation = pn.bind(collect_messages, button_conversation)

dashboard = pn.Column(
    inp,
    pn.Row(button_conversation),
    pn.panel(interactive_conversation, loading_indicator=True, height=300),
)

dashboard
```



In [ ]: