Research Assistant Agent - Final Project

Created by: Jazmine, Mustafa, Olubenga, Kimberly

In today's academic and professional landscape, researchers are often overwhelmed by the sheer volume of information they must sift through. Our final project, the Research Assistant Agent, was designed to solve this problem.

Developed by the Artificial Titans team, this intelligent assistant automates and enhances key aspects of the research process—ranging from information retrieval and citation generation to summarization and adaptive learning.

Built using Azure AI Studio, Python, and web search APIs like SerpAPI, our agent is tailored for students, scholars, and professionals. It can:

- Fetch relevant academic data in real time,
- Format citations in APA, MLA, or Chicago styles,
- Summarize dense material into digestible insights,
- And continuously learn through user feedback using reinforcement learning.

By doing so, the agent not only boosts productivity but also helps reduce human error and cognitive overload.

1. Setup & Imports Load required libraries and configure API keys or settings.

```
#!pip install --upgrade google-search-results
#!pip install --upgrade openai
#!pip install --upgrade azure-core
#!pip install --upgrade requests
#!pip install --upgrade beautifulsoup4
!pip install google-search-results
```

```
Collecting google-search-results
  Downloading google_search_results-2.4.2.tar.gz (18 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from google-search-results) (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->google-search-results
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->google-search-results) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->google-search-results) (2.4
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->google-search-results) (202
Building wheels for collected packages: google-search-results
  Building wheel for google-search-results (setup.py) ... done
  Created wheel for google-search-results: filename=google_search_results-2.4.2-py3-none-any.whl size=32010 sha256=3a07f79b25508624cb024
  Stored in directory: /root/.cache/pip/wheels/6e/42/3e/aeb691b02cb7175ec70e2da04b5658d4739d2b41e5f73cd06f
Successfully built google-search-results
Installing collected packages: google-search-results
Successfully installed google-search-results-2.4.2
```

```
import os, time, datetime, requests, bs4, re, urllib.parse
from typing import List
from openai import AzureOpenAI
from serpapi import GoogleSearch
from openai._exceptions import RateLimitError

os.environ["SERPAPI_KEY"]  = "2ee326c6c0fa9ddc4422160dc9a8a40117e2e0816121b80082c4b098d51141e1"
os.environ["AZURE_OPENAI_KEY"] = "EtQUizZ46ezzyQrDLU1PS8fKJmfbBms17H3cSMXnhnMoqkMpp5sbJQQJ99BDACHYHv6XJ3w3AAAAACOGW4Lu"

# ── API KEYS ──────────────────────────────────────
SERP_API_KEY = os.getenv("SERPAPI_KEY")
AZURE_KEY    = os.getenv("AZURE_OPENAI_KEY")

client = AzureOpenAI(
    api_key=AZURE_KEY,
    api_version="2024-12-01-preview",
    azure_endpoint="https://ai-keremyucedag20168285ai615670493966.openai.azure.com",
)

# ── LLM helper ────────────────────────────────────
def chat(msgs, T=0.7, retries=3):
    for _ in range(retries):
        try:
            r = client.chat.completions.create(
                model="itai-2376", messages=msgs,
```

```
                max_tokens=512, temperature=T, top_p=1.0
            )
            return r.choices[0].message.content
        except RateLimitError as e:
            time.sleep(getattr(e, "retry_after", 5))
    raise RuntimeError("LLM rate-limit")

# ── SEARCH LAYERS ──────────────────────────────
def serp_search(q:str, k:int=3)->List[dict]:
    if not SERP_API_KEY: return []
    try:
        d = GoogleSearch({"q":q,"api_key":SERP_API_KEY,"num":k,"hl":"en"}).get_dict()
        return [{"title":r["title"],"url":r["link"]}
                for r in d.get("organic_results",[])[:k]]
    except Exception:
        return []

DDG = "https://lite.duckduckgo.com/50x/?q={q}"
UA  = ("Mozilla/5.0 (iPhone; CPU iPhone OS 15_0) AppleWebKit/605.1.15 "
       "(KHTML, like Gecko) Version/15.0 Mobile/15E148 Safari/604.1")
def ddg_search(q:str, k:int=3)->List[dict]:
    html=requests.get(DDG.format(q=urllib.parse.quote_plus(q)),
                      headers={"User-Agent":UA},timeout=10).text
    soup=bs4.BeautifulSoup(html,"html.parser")
    links=soup.select("a.result-link--title") or soup.select(".result-link, .result-link--news")
    return [{"title":re.sub(r"\s+"," ",a.get_text(' ',strip=True)),"url":a['href']}
            for a in links[:k]]

def web_search(q:str, k:int=3)->str:
    today=datetime.date.today().isoformat()
    hit=serp_search(q,k)
    if hit:
        print("🔍 SerpAPI hit")
        return "\n".join(f"[{i}] {h['title']} – {h['url']} ({today})"
                         for i,h in enumerate(hit,1))
    hit=ddg_search(q,k)
    if hit:
        print("🌐 DuckDuckGo hit")
        return "\n".join(f"[{i}] {h['title']} – {h['url']} ({today})"
                         for i,h in enumerate(hit,1))
    return f"[1] No results found ({today})"

# ── ReAct agent (SEARCH → SUMMARIZE) ──────────
def research(question:str, max_turns:int=6):
    msgs=[{"role":"system","content":
           ("First output SEARCH:<keywords>. After I return the sources, output "
            "SUMMARIZE:<draft answer>.\n"
            "Use inline citations [1] [2] … and copy the numbered list beneath "
            "a 'Sources:' heading.")},
          {"role":"user","content":question}]
    stage, turns, last_text, sources_block = 0, 0, "", ""

    while True:
        if turns >= max_turns:
            return last_text or "⚠️ No summarized answer after retries."
        resp=chat(msgs); turns+=1; last_text=resp
        first=resp.lstrip().splitlines()[0].strip()

        # ---- SEARCH ----
        if stage==0 and first.startswith("SEARCH:"):
            sources_block = web_search(first[7:].strip())
            msgs[-1:]=[
                {"role":"assistant","content":sources_block},
                {"role":"assistant",
                 "content":("Now output SUMMARIZE:<draft answer> with inline citations "
                            "and include the numbered list under 'Sources:'.")}
            ]
            stage=1; continue

        # ---- SUMMARIZE ----
        if stage==1 and first.startswith("SUMMARIZE:"):
            draft = first[10:].strip()
            if "Sources:" not in draft:               # auto-append if missing
                draft += "\n\nSources:\n" + sources_block
            return draft

        # ---- remind when off-track ----
```

```
        reminder = "Begin with SEARCH:<keywords>." if stage==0 else "Awaiting SUMMARIZE:<draft>."
        msgs.append({"role":"assistant","content":reminder})

## Ask your questions here.
# ── Demo ----------------------------------------
if __name__=="__main__":
    answer = research("Who is George Washington?")
    print("\n✅ FINAL ANSWER:\n", answer)
```

```
🔍 SerpAPI hit

✅ FINAL ANSWER:
George Washington, the first President of the United States, served two terms from 1789 to 1797. Before his presidency, he played a piv

Sources:
[1] George Washington – https://www.thenmusa.org/biographies/george-washington/ (2025-05-08)
[2] George Washington | Life, Presidency, Accomplishments, & ... – https://www.britannica.com/biography/George-Washington (2025-05-08)
[3] Biography of George Washington – https://www.mountvernon.org/george-washington/biography (2025-05-08)
```

```
import ipywidgets as widgets
from IPython.display import display, clear_output
import traceback
import threading
import time
from functools import lru_cache

# Cache for search and LLM results
search_cache = {}
llm_cache = {}

# Wrapper for chat to apply optimizations
def optimized_chat(msgs, T=0.5, retries=3):
    # Cache key based on messages
    cache_key = str(msgs)
    if cache_key in llm_cache:
        return llm_cache[cache_key]

    # Reduce max_tokens for faster response
    for _ in range(retries):
        try:
            r = client.chat.completions.create(
                model="itai-2376",
                messages=msgs,
                max_tokens=256,  # Reduced from 512
                temperature=T,   # Lowered from 0.7
                top_p=1.0
            )
            result = r.choices[0].message.content
            llm_cache[cache_key] = result
            return result
        except RateLimitError as e:
            time.sleep(getattr(e, "retry_after", 2))  # Reduced from 5
    raise RuntimeError("LLM rate-limit")

# Parallel search wrapper
def parallel_search(q: str, k: int = 3):
    cache_key = f"{q}_{k}"
    if cache_key in search_cache:
        return search_cache[cache_key]

    results = []
    def run_serp():
        serp_results = serp_search(q, k)
        if serp_results:
            results.append(("SerpAPI", serp_results))

    def run_ddg():
        ddg_results = ddg_search(q, k)
        if ddg_results:
            results.append(("DuckDuckGo", ddg_results))

    # Run searches in parallel
    threads = [
        threading.Thread(target=run_serp),
        threading.Thread(target=run_ddg)
    ]
```

```
        ]
        for t in threads:
            t.start()
        for t in threads:
            t.join()

        # Format results
        today = datetime.date.today().isoformat()
        if results:
            # Prefer SerpAPI if available, else use DuckDuckGo
            source_type, hit = results[0]
            print(f"🔍 {source_type} hit")
            formatted = "\n".join(f"[{i}] {h['title']} - {h['url']} ({today})"
                                  for i, h in enumerate(hit, 1))
            search_cache[cache_key] = formatted
            return formatted
        formatted = f"[1] No results found ({today})"
        search_cache[cache_key] = formatted
        return formatted

    # Optimized research wrapper
    def optimized_research(question: str, max_turns: int = 6):
        cache_key = question
        if cache_key in search_cache:
            return search_cache[cache_key]

        msgs = [
            {"role": "system", "content":
             ("First output SEARCH:<keywords>. After I return the sources, output "
              "SUMMARIZE:<draft answer>.\n"
              "Use inline citations [1] [2] … and copy the numbered list beneath "
              "a 'Sources:' heading.")},
            {"role": "user", "content": question}
        ]
        stage, turns, last_text, sources_block = 0, 0, "", ""

        with output:
            print("Generating search keywords...")
        while True:
            if turns >= max_turns:
                return last_text or "⚠️ No summarized answer after retries."
            resp = optimized_chat(msgs)
            turns += 1
            last_text = resp
            first = resp.lstrip().splitlines()[0].strip()

            # SEARCH stage
            if stage == 0 and first.startswith("SEARCH:"):
                with output:
                    print("Performing web search...")
                sources_block = parallel_search(first[7:].strip(), 3)
                msgs[-1:] = [
                    {"role": "assistant", "content": sources_block},
                    {"role": "assistant", "content":
                     ("Now output SUMMARIZE:<draft answer> with inline citations "
                      "and include the numbered list under 'Sources:'.")}
                ]
                stage = 1
                continue

            # SUMMARIZE stage
            if stage == 1 and first.startswith("SUMMARIZE:"):
                with output:
                    print("Summarizing results...")
                draft = first[10:].strip()
                if "Sources:" not in draft:
                    draft += "\n\nSources:\n" + sources_block
                search_cache[cache_key] = draft
                return draft

            # Remind if off-track
            reminder = "Begin with SEARCH:<keywords>." if stage == 0 else "Awaiting SUMMARIZE:<draft>."
            msgs.append({"role": "assistant", "content": reminder})

    # Create query input box
    query_box = widgets.Text(
        value='',
        placeholder='Enter your question (e.g., Who is George Washington?)',
```

```
    description='Question:',
    layout={'width': '600px'}
)

# Create submit button
submit_button = widgets.Button(
    description='Submit',
    button_style='success',
    tooltip='Click to submit your question',
)

# Create output area
output = widgets.Output()

# Define button click handler
def on_submit_clicked(b):
    with output:
        clear_output()
        question = query_box.value.strip()
        if not question:
            print("⚠️ Please enter a question.")
            return
        print(f"Processing: {question}")
        try:
            # Debug: Test search functions
            print("Testing SerpAPI...")
            serp_results = serp_search(question, 3)
            print(f"SerpAPI Results: {len(serp_results)} found")

            print("Testing DuckDuckGo...")
            ddg_results = ddg_search(question, 3)
            print(f"DuckDuckGo Results: {len(ddg_results)} found")

            # Run optimized research
            answer = optimized_research(question)
            print(f"\n✅ Answer:\n{answer}")
        except Exception as e:
            print(f"⚠️ Error: {str(e)}")
            print("Stack Trace:")
            print(traceback.format_exc())

# Link button to handler
submit_button.on_click(on_submit_clicked)

# Display widgets
display(query_box, submit_button, output)
```

Question: What is the difference a list and tuple in python?

Submit

```
Processing: What is the difference a list and tuple in python?
Testing SerpAPI...
SerpAPI Results: 3 found
Testing DuckDuckGo...
DuckDuckGo Results: 0 found
Generating search keywords...
Performing web search...
🔍 SerpAPI hit
Summarizing results...

✅ Answer:
Lists and tuples are fundamental Python data structures that store collections of items. Both can store elements of various data types
and can be indexed and iterated, making them versatile for programming tasks. However, lists and tuples have distinct differences that
make them suitable for different uses.

Sources:
[1] Difference Between List and Tuple in Python - https://www.geeksforgeeks.org/python-difference-between-list-and-tuple/ (2025-05-08)
[2] Python Tuples vs. Lists - https://builtin.com/software-engineering-perspectives/python-tuples-vs-lists (2025-05-08)
```