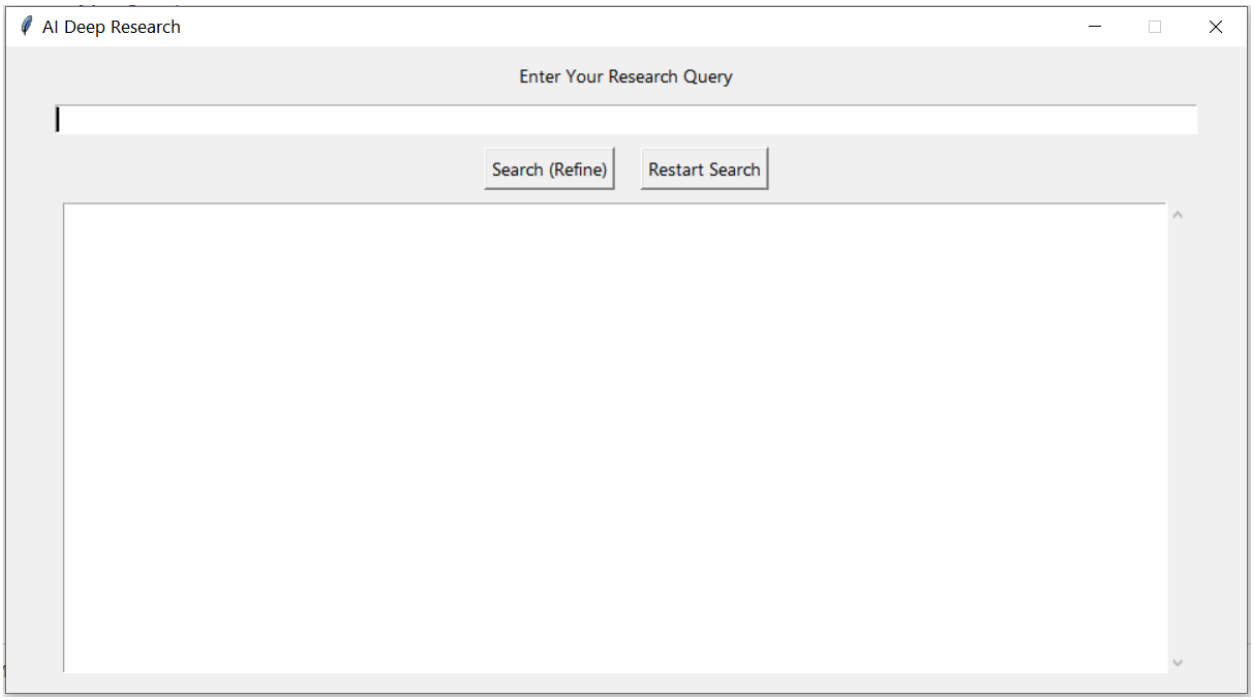
AI Agent-Based Deep Research

A Deep Research AI Agentic System that crawls websites using Tavily for online information gathering. Implement a dual-agent system with one agent focused on research and data collection, while the second agent functions as an answer drafter. The system utilizes the LangGraph & LangChain frameworks to effectively organize the gathered information.

Explanation:

To make the application more user-friendly I gave it GUI appearance (using tkinter), which look like the following:



To search any specific topic user enters his/her query, which is passed to 'StateGraph', which creates state space using langgraph library.

```
55  graph = StateGraph(State)
56  graph.add_node("research", research_agent)
57  graph.add_node("answer", answer_agent)
58  graph.add_edge("research", "answer")
59  graph.add_edge(START, "research")
60
61  graph_compiled = graph.compile()
62
63  state = {"messages": []}
```

Then a research node is added in the graph from where the search starts. Research agent is a function that uses TavilySearchResults() method to search the query by crawling the web and returns a list of results.

```python
tool = TavilySearchResults(max_results=2)
llm = ChatGoogleGenerativeAI(model="gemini-2.0-flash")
llm_with_tools = llm.bind_tools([tool])

class State(TypedDict):
    messages: Annotated[list, add_messages]

def research_agent(state: State):
    last_message = state["messages"][-1]
    query = last_message.content if hasattr(last_message, "content") else str(last_message)

    search_results = tool.invoke(query)

    formatted_result = "Search Results: "
    if search_results:
        for i in search_results:
            formatted_result = formatted_result + i['content']
    else:
        formatted_result = "No relevant search results found."

    return {"messages": state["messages"] + [formatted_result]}
```

The after getting the results from tavily, the search is passed to Google Genai LLM and the answer agent generates a relevant answer using the search. The answer node is added to the graph and results generated by the answer agent is then shown to the user

```python
def answer_agent(state: State):
    response = llm_with_tools.invoke(state["messages"])

    if hasattr(response, "content"):
        response_text = response.content

    return {"messages": state["messages"] + [response_text]}

```

The answer agent keeps track of all the old queries and results too, so that with each new query the results get more refined.

The user will have to option to keep refining old research or restart a new search using this application.

-by Khushi Khandal