# ***Chatvily***

---

**Introduction**

In today's digital era, gathering accurate, relevant, and well-organized information is crucial. We propose a **Deep Research Agentic System** that crawls online data using **Tavily**, processes it using **LangChain** and **LangGraph**, and structures it into a highly professional, clear output.

The system uses a **dual-agent mechanism**:

- **Research Agent**: Searches and collects the best information.
- **Answer Drafting Agent**: Structures, refines, and critiques the final output.

The system ensures that the information is **relevant, fact-checked, clear, and engaging**.

---

**Unique Aspects**

- **Reranker Layer**: Ensures only the most relevant search results are considered.
- **Critic Agent**: Conducts a quality check on clarity, relevance, and factual accuracy.
- **Tone Control**: Maintains a professional yet engaging communication style.
- **Self-Correcting**: If the output fails quality checks, it auto-corrects by re-searching.
- **Scalable**: Can be expanded with additional specialized agents.

---

**Detailed System Explanation**

**1. Setting API Keys**

Securely connect to Tavily (for intelligent web search) and OpenAI (for LLM tasks).

python

CopyEdit

```
os.environ["TAVILY_API_KEY"] = "TAVILY KEY"

os.environ["OPENAI_API_KEY"] = "OPENAI KEY"
```

---

**2. Tool Initialization**

- **TavilySearchResults**: Performs the web search.
- **CrossEncoder**: Reranks results based on relevance.
- **ChatOpenAI**: Formats and critiques the responses.

```
search_tool = TavilySearchResults()

reranker = CrossEncoder('cross-encoder/ms-marco-MiniLM-L-6-v2')

llm = ChatOpenAI(model="gpt-3.5-turbo")
```

---

**3. Research Agent - Search Function**

Uses Tavily to search for user queries.

```
def search_fn(query):

  results = search_tool.invoke({"query": query})

  return results
```

---

**4. Research Agent - Rerank Function**

Pairs the user query with each search result and reranks them based on relevance.

```
def rerank_fn(query, results):

  pairs = [(query, item["content"]) for item in results]

  scores = reranker.predict(pairs)

  reranked = [item for _, item in sorted(zip(scores, results), key=lambda x: x[0], reverse=True)]

  return reranked[:3]
```

---

## 5. Answer Drafting Agent - Formatting Function

Summarizes the best results into a professional, engaging tone.

```
def format_fn(reranked_results):
    prompt = ChatPromptTemplate.from_template("""
Format the following search results into a clean, professional tone.
Make it engaging but formal. Summarize the key points clearly.
{results}
    """)
    formatted = llm.invoke(prompt.format_prompt(results=reranked_results).to_messages())
    return formatted.content
```

---

## 6. Answer Drafting Agent - Critic Function

Checks if the output is clear, professional, consistent, and relevant.

```
def critic_fn(formatted_result):
    prompt = ChatPromptTemplate.from_template("""
Act as a quality critic. Does the following response meet these rules:
- Is it clear and professional?
- Is it factually consistent?
- Is it relevant to the original query?

Respond "PASS" if all good, otherwise "FAIL" with reason.

Response:
{response}
    """)
    critique = llm.invoke(prompt.format_prompt(response=formatted_result).to_messages())
    return critique.content
```

---

**7. Main Logic Controller**

Coordinates the entire workflow: search → rerank → format → critique → final output.
If quality fails, it auto-repeats.

```
def main_logic(query):

  search_results = search_fn(query)

  reranked = rerank_fn(query, search_results)

  formatted = format_fn(reranked)

  critique = critic_fn(formatted)


  if "FAIL" in critique:

    print(" Critic said FAIL. Re-running search...")

    return main_logic(query)

  else:

    print(" Critic said PASS. Final result ready!")

    print(formatted)

    return formatted
```

---

**8. Run Everything**

User enters a query → Full pipeline runs.

```
user_query = input("Enter your search query: ")

final_result = main_logic(user_query)
```

---