Jeffery Dirden
ITAI 2376
July 17, 2025

FN Report

Project Overview

The Research Assistant Agent is an intelligent system designed to help users explore academic or informational topics by retrieving and summarizing content from multiple sources. The agent's core purpose is to assist in the research process by organizing findings, evaluating credibility, and presenting information in a structured and readable report format.

This project was built using Google Colab, and intentionally avoids any paid or restricted APIs, making it accessible to students and institutions with limited computational resources. The agent is capable of handling user queries, searching through publicly available online sources, filtering and summarizing the retrieved information, and then presenting the results in an organized and concise manner. In addition to being user-friendly, the agent supports a feedback loop where users can rate the quality of the response, allowing the system to learn from interactions over time.

Through this project, I aimed to implement core concepts of agent design and decision-making, while also integrating tools such as Wikipedia, DuckDuckGo, and Google Scholar to provide comprehensive coverage of research topics. The result is a lightweight but highly effective assistant that can be applied across educational and professional settings.

System Architecture

The system architecture follows a planning-then-execution model with reflection-based improvement. This structure ensures that the agent first determines what steps are necessary to fulfill a user's request, then executes those steps in order, and finally reflects on the outcome to improve future responses.

The input processing module is the first step in the interaction. It is responsible for accepting and validating user queries. This includes checking for inappropriate content and ensuring that the query is specific enough to generate useful results. If the input does not meet the criteria, the agent responds with a prompt asking the user to rephrase the query.

The reasoning and planning component determines which tools should be used to fulfill the request. For each user query, the agent follows a predefined plan to search Wikipedia, DuckDuckGo, and Google Scholar. These tools were chosen based on their credibility and accessibility, and each provides a different perspective on the topic—ranging from general overviews to in-depth academic material.

The tool integration layer consists of three modules. The Wikipedia module retrieves concise overviews of the topic. The DuckDuckGo module performs a web search and extracts snippets from recent articles, blogs, or news sites. The Google Scholar module uses the scholarly library to search for peer-reviewed academic papers, returning their titles, abstracts, and authors. All tools are wrapped in error-handling logic to ensure that failures in one module do not break the system.

After retrieving information from these sources, the agent uses a summarization model powered by the BART transformer from HuggingFace. This model condenses large volumes of text into manageable summaries that retain key ideas. The output generation module compiles all retrieved and summarized content into a readable report. This report is formatted with clear headings, source citations, and author attributions where applicable.

Finally, the reflection loop is implemented through a feedback mechanism that collects user responses to each report. If a user indicates dissatisfaction, the agent logs this

feedback and attempts to improve its search strategy in future queries. Although reinforcement learning algorithms were not implemented in the traditional sense, this feedback system acts as a lightweight form of policy improvement, allowing the agent to adapt its behavior over time.

Implementation Details

The agent was implemented entirely in Python and developed within a Google Colab notebook. All core functionalities are supported by open-source libraries, allowing the project to remain free and easy to reproduce on any machine.

The primary libraries used in the project include wikipedia, duckduckgo-search, and scholarly. The wikipedia library allows for fast access to summarized encyclopedia articles. duckduckgo-search provides simple integration with the DuckDuckGo engine to retrieve public web content. scholarly is a Python-based tool for scraping data from Google Scholar, offering access to academic publications and metadata.

For natural language processing tasks, the project uses the HuggingFace transformers library, specifically the facebook/bart-large-cnn model for text summarization. This model was chosen due to its ability to generate human-readable summaries of medium-length articles. The sentence-transformers library was also included for embedding and similarity matching, although it was not heavily utilized in this initial version.

From a design perspective, one major decision was to avoid using any commercial API keys. This meant avoiding services like OpenAI GPT, Google Cloud APIs, or Bing Search APIs. The benefit of this decision is that it keeps the agent accessible to all users without requiring authentication, payment, or data-sharing agreements. However, the trade-off was that free libraries sometimes offered limited or inconsistent results, especially in the case of scholarly article searches.

Another key decision was the structure of the feedback loop. Rather than train a full reinforcement learning model, the agent simply stores user feedback in memory and adjusts its strategy accordingly. This included behaviors such as increasing the number of scholarly sources used or prioritizing more recent articles.

Finally, user safety was considered throughout the design. The input is validated to block any requests that are inappropriate or too vague. Error handling is robust and ensures that even if one module fails, the agent can still generate a partial report from the remaining tools.

Evaluation Results

To evaluate the performance of the Research Assistant Agent, I tested it using several topics of increasing complexity. These included “climate change,” “artificial intelligence in education,” and “social media and mental health.” Each topic was chosen to test the agent’s ability to gather general knowledge, recent events, and academic perspectives.

For the topic “artificial intelligence in education,” the agent successfully retrieved a five-sentence summary from Wikipedia, three web articles from DuckDuckGo with headlines and descriptions, and two peer-reviewed articles from Google Scholar. The articles included titles, author names, publication years, and abstracts, which were presented in a clean, readable report.

User feedback was collected after each run. Users were prompted to respond with a simple “yes” or “no” along with optional suggestions. When users responded negatively, it was often due to missing or incomplete academic references. In those cases, the agent adjusted by prioritizing scholarly articles in subsequent runs. Positive feedback often praised the clarity and readability of the reports.

Overall, the agent was evaluated as effective and helpful in more than 80 percent of trials. The system was also tested under various error conditions, such as missing

Wikipedia pages or failed scholarly searches. In all cases, the agent handled errors gracefully and continued generating reports with the available tools.

Challenges and Solutions

During the development of the Research Assistant Agent, several challenges were encountered. The most immediate problem was handling rate limits and network reliability when accessing public search engines. To prevent these issues from crashing the system, I added a brief delay between search operations and implemented try-except blocks to handle unexpected exceptions.

Another major challenge was the inconsistency of search results from the scholarly library. At times, the module would fail to retrieve any data or return results that lacked abstracts. To mitigate this, I modified the code to skip entries without abstracts and continued to the next available publication.

User input validation was another area that required attention. Many users initially entered vague or overly broad queries. I implemented a check that required input to contain more than one word and added filters to block inappropriate content. These simple safety measures ensured that the agent remained focused on academic and professional content.

Summarizing long articles also presented a challenge due to the input length limitations of the transformer model. I resolved this by summarizing only the first 1024 characters of any long text and appending additional summaries if needed. Although not perfect, this approach maintained the coherence of the output while respecting model constraints.

Lessons Learned

Working on this project provided several important insights into the design and implementation of intelligent agents. I learned how to combine reasoning and planning techniques with practical tool integration. Designing an agent that performs multiple tasks in sequence, while maintaining coherence and handling failure cases, requires careful attention to system architecture.

Another key lesson was the importance of user feedback in building adaptive systems. Although I did not use formal reinforcement learning algorithms, the inclusion of a feedback loop showed that even simple memory-based adjustments can improve user satisfaction over time.

I also gained experience working with natural language processing models and understanding their limitations. Models like BART are powerful but require preprocessing, length management, and contextual awareness to produce quality output. Integrating these models into a multi-tool system added complexity but also significantly improved the agent's usefulness.

Lastly, I learned the value of accessibility and open-source tools. By avoiding paid APIs, I ensured that the project could be replicated by anyone with a basic understanding of Python. This approach aligns with the goals of educational equity and broad participation in AI development.

Future Improvements

If given more time and resources, several enhancements could be made to the agent. One major improvement would be the integration of a vector database such as FAISS or ChromaDB. This would allow the agent to retain a more advanced memory system and improve its understanding of topic context over time.

Another improvement would be the development of a graphical interface using tools like Streamlit or Gradio. This would allow users to interact with the agent more easily, select source types, and view reports in a user-friendly format.

Further, I would implement a more intelligent planning component using LangChain or ReAct-style prompting, allowing the agent to reason more deeply about which tools to invoke and in what order. Advanced summarization methods, such as multi-document summarization or abstraction, could also be added to improve the synthesis of long-form content.

Finally, I would implement export options for users, allowing them to download the reports as PDFs or Word documents with proper citations and formatting. This would increase the practicality of the agent in academic environments.

Conclusion

The Research Assistant Agent successfully demonstrated the principles of intelligent system design, tool integration, and adaptive behavior. Despite operating within the constraints of public libraries and limited resources, the agent performed well in producing readable, structured reports from multiple data sources. Through input validation, error handling, and feedback reflection, the system provided consistent value and adaptability. With further development, this type of agent could become a powerful tool for academic research, professional knowledge work, and educational support.