As part of the Intelligent Agents module, a Google Scholar Research Assistant Application has been developed. This is a desktop-based intelligent agents that will allow researchers to search for academic literature using search terms and optional year filters. In particular, this agent retrieves structured results from Google Scholar and exports them to SCV for subsequent offline analysis.

During this presentation, as described in the slide, it will be presented the process of developing an Intelligent Agent App, including a full step-by-step description of the steps followed to develop the app, the testing steps applied, the creation of the executable, and the app functionality. Additionally, challenges faced during the process will be presented, as well as conclusions and relevant references.

This project aims to assist researchers by streamlining the process of retrieving academic publications from Google Scholar. Instead of manually searching and copying results, this tool provides:

- A friendly desktop interface via Streamlit.
- Automatic integration with Google Scholar through SerpApi.
- The ability to filter results by year.
- Exporting results in CSV format for further analysis.

Our goals were simplicity, performance, and portability. The final product is an executable `.exe` file that runs the app locally.

The project uses Python and key libraries like Streamlit, SerpApi, Pandas, and BeautifulSoup. In particular:

SerpApi was used to retrieve results from Google Scholar in a structured way. By selecting this library, compliance with anti-scraping policies was ensured, preventing any restriction on automated data extraction;

BeautifulSoup was used for parsing HTML documents when needed. BeautifulSoup was chosen over other libraries because it is lightweight, easy to implement, and better suited for handling static HTML content (Kumar et al., 2024; Mitchell, 2018);

Pandas was used for the data structure and storage; and

Streamlit was used for building the web-based GUI for the system.

It is worth also to be mentioned that PyInstaller was used to bundle the Python application into a standalone Desktop executable.

Now, looking at the development and testing code. The core functionality lies in the fetch_scholar_results function. In particular, this function takes a search query and a publication year as input; the latter is optional to each researcher. It then sends a request to Google Scholar via SerpApi, querying the database with the researcher's search terms and filters. Following the request, the response from Google Scholar is processed to extract key information such as the article title, snippet, link, authors, and publication year. This data is organized into a Pandas DataFrame, making it easily manageable for subsequent offline analysis.

Moving on to the user interface, which is built using Streamlit. The UI is designed to be intuitive and easy to use. Users can input their search query and specify a publication year should they wish to. Upon clicking the search button, the app calls the fetch_scholar_results function to retrieve the relevant results. Then the results are displayed in a neatly formatted table directly within the web app. This allows users to quickly browse through the findings and assess their relevance. Additionally, the app includes a download button that researchers can use in order to download the results as a CSV file.

To ensure the reliability and accuracy of the app, I incorporated unit tests using the unittest framework. These tests simulate various scenarios and validate the functionality

of different parts of the code. Mocking is employed to isolate dependencies and ensure that the tests focus solely on the app's logic. As you can see, the results show that the app works well.

Next step was the creation of the executable file so to be able to access the application by a double click. In this slide the process followed is presented.

Now going to an example of the application. Entering the application the user can type the query he/she wants and the year; the latter as already said is optional. In this example what we were looking for was articles in Google scholar for “fraud prevention and machine learning” during 2024. By clicking on “Search” button, the application created a table of the results, including information regarding the author, title, snippet, link to the article, and year. Then the user has either the option to go through the application and by using the link provides be directed to the article or by clicking on “Download” button, download the list in a structured CSV for subsequent analysis.

To sum up, this Google Scholar Search App provides an efficient and effective way for researchers to quickly access academic literature. By automating the search process and providing a user-friendly interface, this tool empowers researchers to focus on their work rather than spending excessive time on manual searches.

Last slide shows the references used for the development of the Agent.

Thank you.

References

Kumar, S., Varha, V., Sharup, Y., Ahamd, S., & Chandy, K. (2024). Leveraging Python for Web Scraping and Data Analysis: Applications, Challenges, and Future Directions. Frontiers in Collaborative Research, 2(1), 65-73.

Massimino, B. (2016). Accessing Online Data: Web-Crawling and Information-ScrapingTechniques to Automate the Assembly of Research Data. Journal of Business Logistics, 37(1), 34-42.

Mitchell, R. (2018) Web Scraping with Python: Collecting More Data from the Modern Web. Second edition. Sebastopol: O’Reilly Media, Incorporated.

VanderPlas, J. (2022). Python Data Science handbook. O'Reilly Media, Inc.

Wooldridge, M. (2009) ‘An Introduction to Multi-Agent Systems’. United Kingdom: John Wiley & Sons, Incorporated.