Assignment topic: Intelligent Agent in Action

Master in Artificial Intelligence

Module: Intelligent Agents

Table of content

1. Introduction

- This application allows researchers to search Google Scholar through a simple desktop interface.

- It uses Streamlit for the interface, SerpApi for fetching results, Pandas for data handling, and BeautifulSoup (if needed) for parsing.

2. Requirements

- - Streamlit: for building the UI
- - SerpApi: structured access to Google Scholar
- - Pandas: for organizing and exporting data
- - BeautifulSoup: for parsing HTML if needed
- - PyInstaller: to create the .exe file
- - unittest & mock: for testing

3. App. Development (1/6)

```
!pip install streamlit
!pip install serpapi
!pip install pandas
!pip install beautifulsoup4
!pip install google-search-results
```

```
   # Import necessary libraries
import streamlit as st
import pandas as pd
•   from serpapi import GoogleSearch
```

3. App. Development (2/6)

```
    data = []
    for item in results.get("organic_results", []):
        data.append({
            "Title": item.get("title"),
            "Snippet": item.get("snippet"),
            "Link": item.get("link"),
            "Authors": item.get("publication_info", {}).get("summary",
"N/A"),
            "Year": year if year else "Unknown",
        })

        return pd.DataFrame(data)

    except Exception as e:
    st.error(f"Error: {str(e)}")
    return pd.DataFrame()
```

3. App. Development(3/6)

```
# Function to fetch Google Scholar results
def fetch_scholar_results(query, year):
    api_key = "my key" # my personal key was used for running the script
   but omitted to be included

        params = {
         "engine": "google_scholar",
         "q": query,
         "api_key": api_key,
    }

        if year:
         params["after_year"] = year  # Filter by publication year

        try:
         search = GoogleSearch(params)
         results = search.get_dict()
```

3. App. Development (4/6)

```
        data = []
    for item in results.get("organic_results", []):
        data.append({
            "Title": item.get("title"),
            "Snippet": item.get("snippet"),
            "Link": item.get("link"),
            "Authors": item.get("publication_info", {}).get("summary",
"N/A"),
            "Year": year if year else "Unknown",
        })

        return pd.DataFrame(data)

    except Exception as e:
    st.error(f"Error: {str(e)}")
    return pd.DataFrame()
# Streamlit UI
st.title("📚 Google Scholar Search")
```

3. App. Development (5/6)

```
# User Input
query = st.text_input("Enter Search Term:")
year = st.text_input("Enter Year (Optional):")

   if st.button("Search"):
    if not query:
        st.error("Please enter a search term.")
    else:
        st.info("Fetching results...")
        df = fetch_scholar_results(query, year)

            if df.empty:
             st.warning("No results found. Try different
   keywords.")
        else:
             st.success(f"Found {len(df)} results!")
             st.dataframe(df)  # Show results in a table
```

3. App. Development (6/6)

```
    # Download button
csv = df.to_csv(index=False).encode("utf-8")
st.download_button(
    label="📥 Download CSV",
    data=csv,
    file_name="scholar_results.csv",
    mime="text/csv",
)
```

4. App. Testing (1/3)

```
import unittest
from unittest.mock import patch, MagicMock
import pandas as pd

class TestGoogleScholarApp(unittest.TestCase):

    @patch("serpapi.GoogleSearch.get_dict")  # Mock API call
 def test_fetch_scholar_results(self, mock_get_dict):
     """Test fetching Google Scholar results with a valid search
term."""
     # Simulated API Response
     mock_get_dict.return_value = {
         "organic_results": [
             {"title": "AI Paper", "snippet": "Abstract text", "link":
"https://example.com",
              "publication_info": {"summary": "Author 1, Author 2"}},
             {"title": "Machine Learning", "snippet": "ML research",
"link": "https://ml.com"}
         ]
     }
```

4. App. Testing (2/3)

```
        search_results = fetch_scholar_results("AI research", "2024")

        self.assertEqual(len(search_results), 2)
        self.assertEqual(search_results.iloc[0]["Title"], "AI Paper")

    def test_process_results(self):
        """Test processing API response into a DataFrame."""
        raw_data = [
            {"Title": "Paper 1", "Snippet": "Text", "Link":
"https://paper1.com", "Authors": "Author A", "Year": "2023"},
            {"Title": "Paper 2", "Snippet": "Text", "Link":
"https://paper2.com", "Authors": "Author B", "Year": "2024"}
        ]
        df = pd.DataFrame(raw_data)

        self.assertIsInstance(df, pd.DataFrame)
        self.assertEqual(df.shape, (2, 5))
```

4. App. Testing (3/3)

```
@patch("pandas.DataFrame.to_csv")  # Mock saving to CSV
    def test_save_to_csv(self, mock_to_csv):
        """Test saving results to CSV without actual file creation."""
        df = pd.DataFrame([{"Title": "Paper", "Link":
   "https://paper.com"}])
        df.to_csv("test_results.csv", index=False)

            mock_to_csv.assert_called_once()  # Check that the method was
   called once

   # Run tests
unittest.TextTestRunner().run(unittest.TestLoader().loadTestsFromTestCase(T
estGoogleScholarApp))
```

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.012s

OK
<unittest.runner.TextTestResult run=3 errors=0 failures=0>
```

5. Creation of executable

Step 1

- Install PyInstaller:

pip install pyinstaller

Step 2

- Run from script folder:

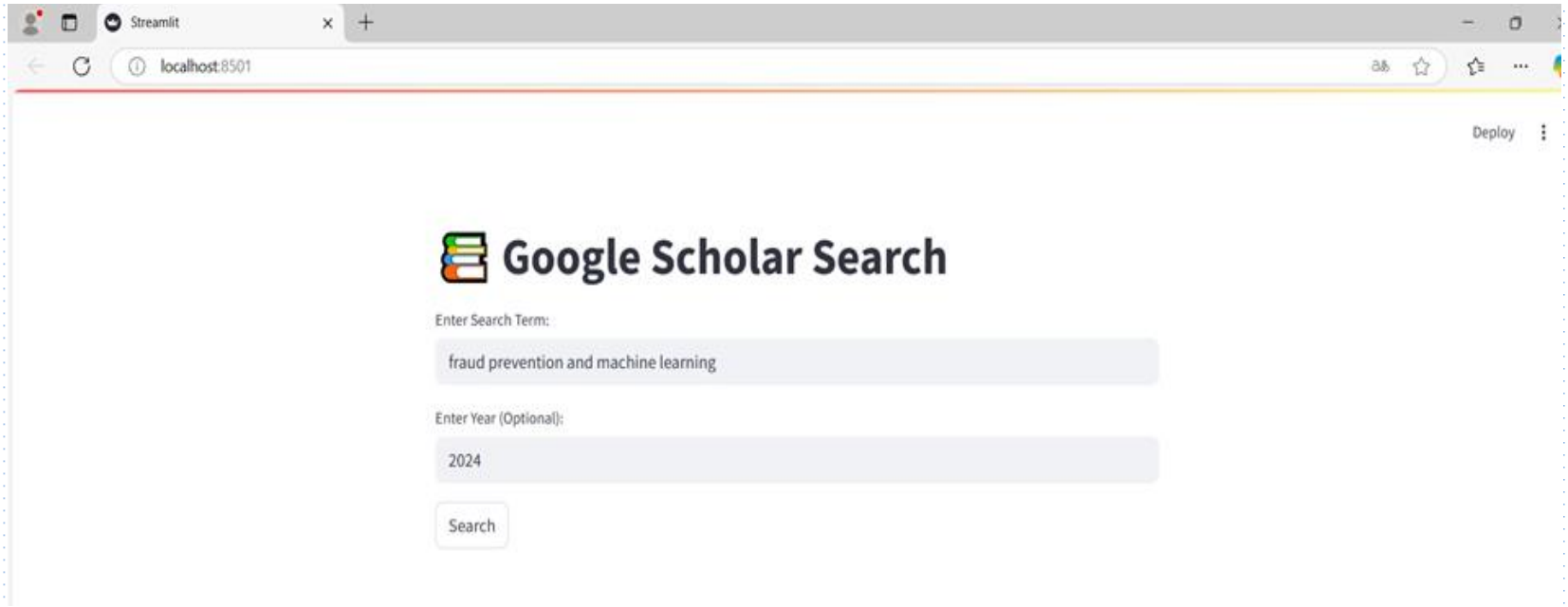
pyinstaller --onefile --noconsole --copy-metadata streamlit --collect-submodules serpapi google_scholar_app.py

Outcome

- Executable created in 'dist/' folder

6. Functionality (1/2)

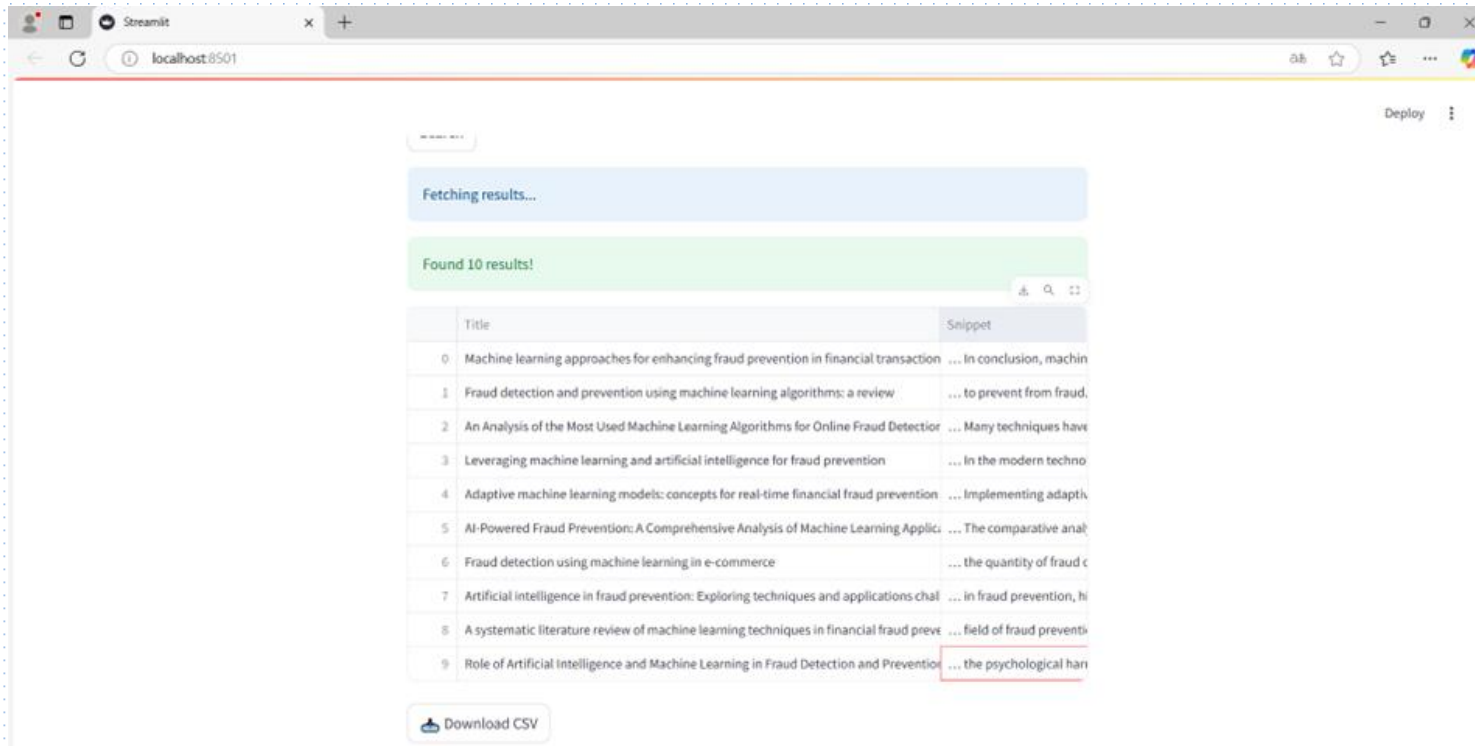
- Enter keywords and optional year



- Click 'Search' to query Google Scholar

6. Functionality (2/2)

- Results shown in a scrollable table



- Download results as CSV with one click

8. Conclusion

- App based on Python code
- App fetches and stores academic results efficiently
- Useful for researchers and students

9. 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.