Automating Research Paper Scraping, Categorization, and Analysis with Python & AI

Introduction

Research in AI and machine learning is expanding rapidly, with thousands of papers published annually in top conferences like NeurIPS. Manually categorizing and extracting relevant research papers can be tedious. In this blog post, I explain how I automated the process using Python, asynchronous web scraping, and AI-based classification with Gemini.

Overview of the Approach

The project consists of two main scripts:

1. `1.py` - Scrapes research paper details (titles, abstracts, and PDF links) from NeurIPS.
2. `2.py` - Extracts text from PDFs, classifies papers into predefined categories, and stores structured results.

The overall workflow is:

1. Scrape paper details from NeurIPS.
2. Extract PDF links and download the research papers.
3. Extract the title and abstract from each paper.
4. Use an AI model to generate research categories.
5. Categorize each paper based on its title and abstract.
6. Store results in a structured JSON format for further analysis.

Implementation Details

1. Web Scraping Using Asynchronous Requests

To efficiently scrape large datasets, I used `aiohttp` with `BeautifulSoup`. The script first fetches the list of papers from NeurIPS and then extracts individual paper details, including their PDF links.

Key Challenges:

- Handling network timeouts and retries.
- Maintaining concurrency with `asyncio.Semaphore` to limit excessive requests.

- Parsing structured data from HTML while avoiding broken links.

Example snippet for fetching NeurIPS papers:

```
async def fetch_page(session, url):
    async with semaphore:
        try:
            async with session.get(url, timeout=30) as response:
                return await response.text()
        except Exception as e:
            print(f"Error fetching {url}: {e}")
            return None
```

After retrieving the paper list, we extract paper titles and links, storing them in `papers_{year}.json`.

---

2. Extracting PDF Links and Downloading Papers

Once paper links are obtained, another async function retrieves the actual PDF links from the paper pages. Then, PDFs are downloaded and stored locally.

Challenges:

- Some papers don't have accessible PDFs.
- Ensuring filename sanitization to avoid OS conflicts.

```
def sanitize_filename(filename):
    return re.sub(r'[<>:"/\\|?*]', '_', filename)
```

This ensures safe file storage when saving PDFs.

---

3. Extracting Text from PDFs

Using `PyMuPDF (fitz)`, we extract the first page of each paper to obtain the title and abstract. This avoids processing the entire document, saving computational resources.

```
def extract_text_from_first_page(pdf_path):
    doc = fitz.open(pdf_path)
    first_page = doc.load_page(0)
    text = first_page.get_text()
    doc.close()
    return text
```

---

4. AI-Based Categorization with Gemini

Once the titles and abstracts are extracted, we use Google's Gemini API to generate research categories dynamically. Another AI model is then used to classify each paper into these categories.

Key Challenges:

- Ensuring AI-generated categories are relevant.
- Handling inconsistent LLM responses (e.g., missing papers in output).

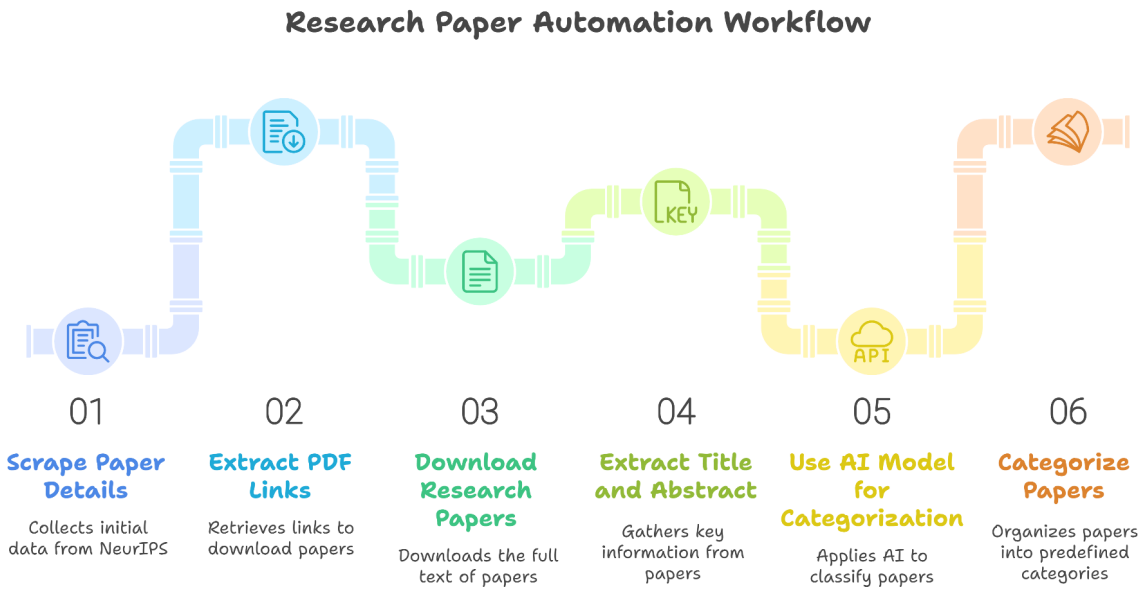
Snippet for AI-based categorization:

```
response = client.models.generate_content(
    model="gemini-2.0-flash",
    contents=f"Identify five common research categories from the following titles and abstracts..."
)
```

To avoid mismatches, we apply regex-based parsing:

```
category_lines = re.findall(r"\d+\.\s\[(.+?)\]", response_text)
classified_categories = [line.split(", ") for line in category_lines]
```

If mismatches occur, missing papers are labeled as "Uncategorized."

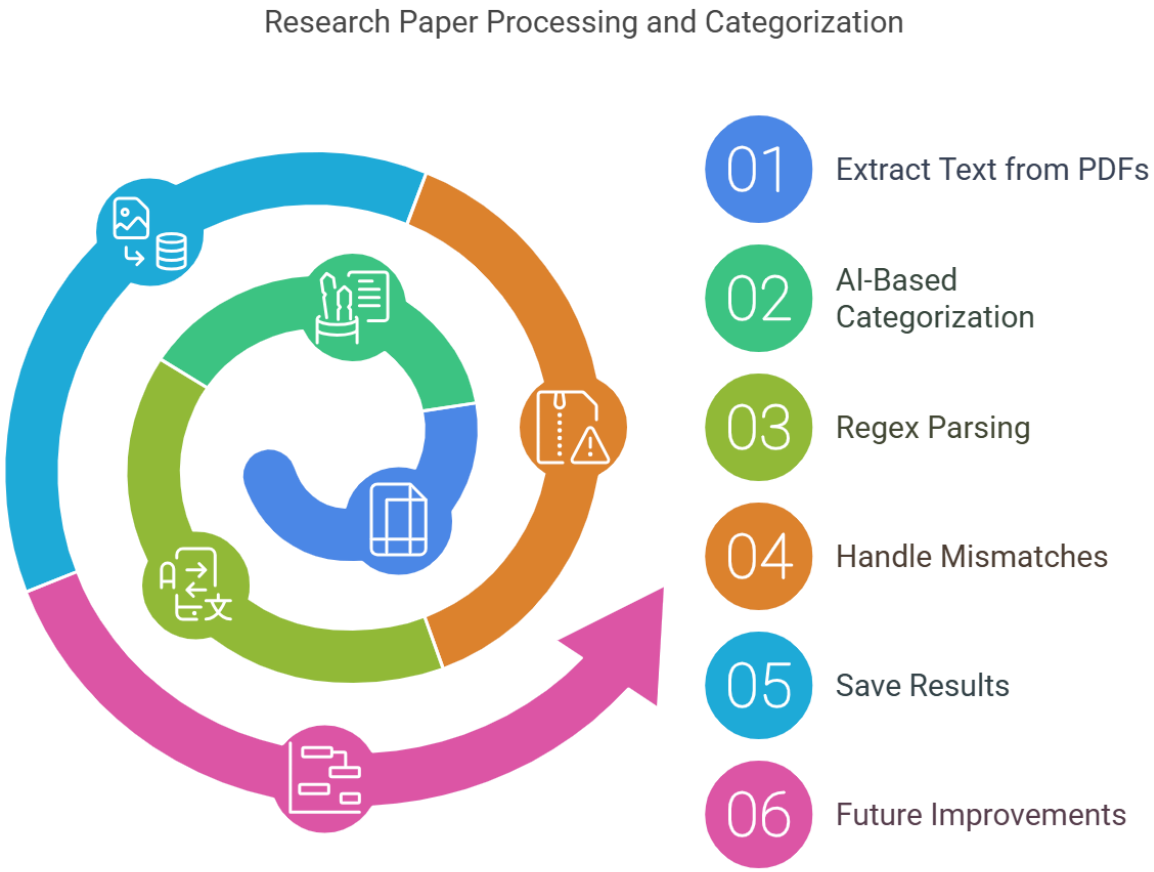


Final Results & Storage

The final categorized papers are saved in `pdf_links_{year}_updated.json`, including their extracted titles, abstracts, and assigned research categories.


```
{
   "title": "Deep Learning for Optimization",
   "pdf_link": "https://papers.nips.cc/paper/2021/file/sample.pdf",
   "categories": ["Deep Learning", "Optimization"]
}
```


This structured dataset can now be used for further analysis, such as trend detection in AI research.



Lessons Learned

- **Asynchronous scraping** significantly speeds up web requests, reducing wait time.
- **Retry strategies** are essential when scraping large datasets due to network inconsistencies.
- **AI classification** is powerful but requires validation to ensure output correctness.
- **File handling best practices** prevent errors when dealing with numerous downloads.

Future Improvements

- Use **LLM fine-tuning** for more accurate categorization.
- Implement **database storage** for better query capabilities.
- Extend scraping to **other conferences** like ICML, CVPR, and ICLR.

This project showcases how automation can streamline research analysis. If you found this useful, feel free to reach out and share your insights!