Table of Contents

[Automating Research Paper Annotation Using Gemini API and Python 2](#_Toc190451064)

[The Challenge 2](#_Toc190451065)

[My Approach 2](#_Toc190451066)

[Step 1: 2](#_Toc190451067)

[Step 2: 2](#_Toc190451068)

[Step 3: 2](#_Toc190451069)

[Step 4: 3](#_Toc190451070)

[Challenges 4](#_Toc190451071)

[Faced Quota and Rate Limits 4](#_Toc190451072)

[Endpoint Configuration 4](#_Toc190451073)

[Data Consistency 4](#_Toc190451074)

[Conclusion 4](#_Toc190451075)

# Automating Research Paper Annotation Using Gemini API and Python

In this post, I’ll share my experience of creating an end-to-end automation pipeline to annotate NeurIPS research papers using Large Language Models (LLMs). I’ll guide you through my methodology, the implementation specifics, and the obstacles I encountered along the way.

## The Challenge

Research papers are published at an astonishing rate, making it difficult for researchers to stay updated with new trends. I aimed to simplify the process of categorizing these papers by automating data annotation. The objective was to classify each paper into one of five categories—Deep Learning, Computer Vision, Reinforcement Learning, NLP, and Optimization—leveraging the capabilities of LLMs, particularly through Google’s Gemini API.

## My Approach

### Step 1:

Scraping the Data I began by developing a web scraper in Python to gather metadata (title, authors, abstract, PDF URL) from the NeurIPS website. The scraper was built in Google Colab to take advantage of its cloud resources, and it downloads the scraped research papers and metadata as a ZIP file.

### Step 2:

Data Extraction and Preparation After downloading the ZIP file to my local machine, I extracted the CSV file containing the metadata. This CSV served as the input for the annotation process. It was essential to ensure that the CSV included the necessary columns—especially the title and abstract of each paper.

### Step 3:

Automating Annotation with Gemini API For the annotation phase, I transitioned to Visual Studio Code. I integrated the Google GenAI library, which allows access to the Gemini API. My Python script utilizes the Gemini API to classify each paper based on a well-crafted prompt. For instance, the prompt instructs the API:

"Classify the following research paper into one of these categories: Deep Learning, Computer Vision, Reinforcement Learning, NLP, Optimization. If it does not clearly fit into any of these, reply with 'Other'."

The script processes a CSV file by sending each paper’s title and abstract to the Gemini API, then adds the returned category to a new column. To avoid rate-limiting issues, a 2-second delay is implemented after each API call.

### Step 4:

Storing the Annotations Once processing is complete, the updated CSV features an additional Category column with the annotations. This organized dataset is now prepared for further analysis or integration into other projects.

1. Implementation Details Here’s a brief overview of the main components of my implementation:

#### Web Scraper:

Developed using Python libraries such as requests and BeautifulSoup, the scraper gathers research papers from NeurIPS and stores metadata either locally or in the cloud.

#### Annotation Script:

The script utilizes the google-genai library. I configured the Gemini API client, created a prompt for categorization, and looped through each record in the CSV to annotate the papers.

Delay for Rate Limiting:

To manage API rate limits, I incorporated a 2-second delay after each API call using time.sleep(2).

#### Error Handling:

The script features try-except blocks to handle API errors, ensuring that problems like quota exhaustion or network issues do not disrupt the entire process.

## Challenges

### Faced Quota and Rate Limits

One significant challenge was reaching the Gemini API quota. Initially, I faced RESOURCE\_EXHAUSTED errors due to excessive requests in a short timeframe. Adding a delay between API calls was essential to mitigate this issue, although it did increase the overall processing time.

### Endpoint Configuration

Another difficulty was making sure that the Gemini API endpoint and model parameters were set up correctly. The official documentation was crucial for establishing the right endpoint (gemini-2.0-flash) and ensuring the payload complied with the API requirements.

### Data Consistency

It was also important to verify that the scraped CSV file included all necessary columns (title and abstract). I had to conduct additional data validation to prevent issues during the annotation process.

## Conclusion

This project showcases the integration of LLMs like Gemini into a data pipeline for automating the annotation of research papers. By utilizing web scraping, data extraction, and API-based classification, I developed a workflow that not only saves time but also lays the groundwork for more in-depth research analysis.

I hope this post encourages you to consider similar automation projects in your own work.