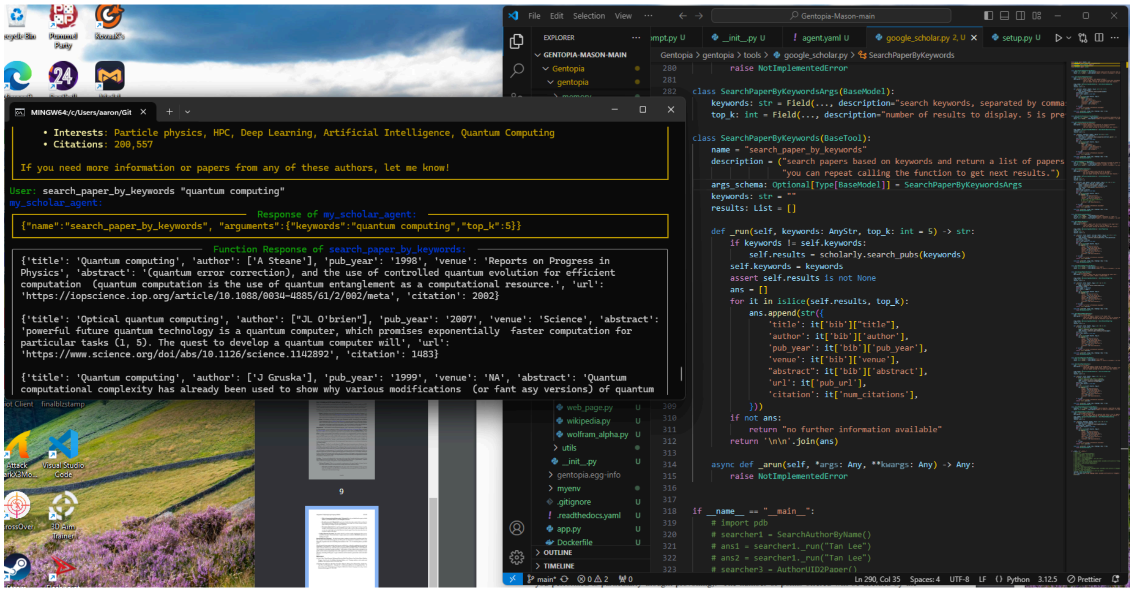
Question 1:
In this project, I built an LLM agent that operates with LLM libraries (specifically, Google Scholar) to help with academic research. The agent performs functions like searching for authors and papers based on specific keywords or author names. This agent is powered by the GPT-4o-mini model and utilizes the Gentopia-Mason framework. The function I personally incorporated into the framework is “SearchPaperByKeywords”. This widget emphasizes on finding papers by specific keywords/phrases.

For instance, it can search for authors by name (e.g., "Nikola Tesla") or find papers related to keywords (e.g., "quantum computing"). This can be a useful tool for researchers who need quick access to scholarly articles and academic resources without manually searching multiple databases.

Question 2:
This agent is important because it helps automate the process of finding relevant research materials, saving time and effort for researchers. For example, it can help academic professionals quickly gather papers related to a specific field like quantum computing, or track down authors working on similar topics. In real life, this type of agent can be integrated into academic search engines, databases, or as a personal research assistant to help researchers stay up-to-date with the latest literature.

Question 3:
To implement the agent, I used the Gentopia-Mason framework to build the LLM agent. This agent can interface with external tools to retrieve information from Google Scholar.

search_paper_by_keywords: This function enables the agent to search for academic papers related to specific keywords (e.g., "quantum computing") and return information like the title, authors, publication year, and citation count.



```
MINGW64:/c/Users/aaron/Git

    • Interests: Particle physics, HPC, Deep Learning, Artificial Intelligence, Quantum Computing
    • Citations: 200,557

 If you need more information or papers from any of these authors, let me know!

User: search_paper_by_keywords "quantum computing"
my_scholar_agent:
                              Response of my_scholar_agent:
 {"name":"search_paper_by_keywords", "arguments":{"keywords":"quantum computing","top_k":5}}

                      Function Response of search_paper_by_keywords:
 {'title': 'Quantum computing', 'author': ['A Steane'], 'pub_year': '1998', 'venue': 'Reports on Progress in
 Physics', 'abstract': '(quantum error correction), and the use of controlled quantum evolution for efficient
 computation  (quantum computation is the use of quantum entanglement as a computational resource.', 'url':
 'https://iopscience.iop.org/article/10.1088/0034-4885/61/2/002/meta', 'citation': 2002}

 {'title': 'Optical quantum computing', 'author': ["JL O'brien"], 'pub_year': '2007', 'venue': 'Science', 'abstract':
 'powerful future quantum technology is a quantum computer, which promises exponentially  faster computation for
 particular tasks (1, 5). The quest to develop a quantum computer will', 'url':
 'https://www.science.org/doi/abs/10.1126/science.1142892', 'citation': 1483}

 {'title': 'Quantum computing', 'author': ['J Gruska'], 'pub_year': '1999', 'venue': 'NA', 'abstract': 'Quantum
 computational complexity has already been used to show why various modifications  (or fant asy versions) of quantum
```

```python
        raise NotImplementedError

class SearchPaperByKeywordsArgs(BaseModel):
    keywords: str = Field(..., description="search keywords, separated by comma
    top_k: int = Field(..., description="number of results to display. 5 is pre

class SearchPaperByKeywords(BaseTool):
    name = "search_paper_by_keywords"
    description = ("search papers based on keywords and return a list of papers
                   "you can repeat calling the function to get next results.")
    args_schema: Optional[Type[BaseModel]] = SearchPaperByKeywordsArgs
    keywords: str = ""
    results: List = []

    def _run(self, keywords: AnyStr, top_k: int = 5) -> str:
        if keywords != self.keywords:
            self.results = scholarly.search_pubs(keywords)
        self.keywords = keywords
        assert self.results is not None
        ans = []
        for it in islice(self.results, top_k):
            ans.append(str({
                'title': it['bib']["title"],
                'author': it['bib']['author'],
                'pub_year': it['bib']['pub_year'],
                'venue': it['bib']['venue'],
                "abstract": it['bib']['abstract'],
                'url': it['pub_url'],
                'citation': it['num_citations'],
            }))
        if not ans:
            return "no further information available"
        return '\n\n'.join(ans)

    async def _arun(self, *args: Any, **kwargs: Any) -> Any:
        raise NotImplementedError


if __name__ == "__main__":
    # import pdb
    # searcher1 = SearchAuthorByName()
    # ans1 = searcher1._run("Tan Lee")
    # ans2 = searcher1._run("Tan Lee")
    # searcher3 = AuthorUID2Paper()
```

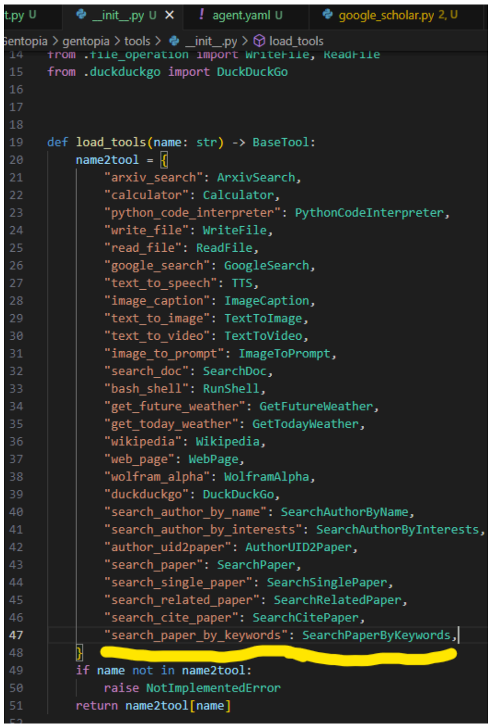
For example, when the query "search_paper_by_keywords 'quantum computing'" is given, the agent will return a list of relevant papers. Here's a snippet of the agent's response:

1. Quantum computing
   - Author: A Steane
   - Published Year: 1998
   - Venue: Reports on Progress in Physics
   - Citations: 2002
   - [Link to Paper](https://iopscience.iop.org/article/10.1088/0034-4885/61/2/002/meta)

Question 4:
While building the agent, I faced a few challenges integrating the Google Scholar API with the language model, especially in handling the structure of the data returned. However, the GPT-4o-mini model proved to be effective in understanding and structuring the results.

Most of the problems I encountered emerged from a lack of download modules/libraries. I was often caught in the wrong directory and editing the wrong files because the README.md file was entirely specific. For instance, I found out about the __init__.py error after running the assemble.py function on scholar. I realized I forgot to add my custom function into load_tools.

If I had more time, I would focus on improving the agent's ability to summarize papers and offer more in-depth analysis. Additionally, integrating more advanced citation tracking and analysis would be an exciting feature to explore.



```python
from .file_operation import WriteFile, ReadFile
from .duckduckgo import DuckDuckGo


def load_tools(name: str) -> BaseTool:
    name2tool = {
        "arxiv_search": ArxivSearch,
        "calculator": Calculator,
        "python_code_interpreter": PythonCodeInterpreter,
        "write_file": WriteFile,
        "read_file": ReadFile,
        "google_search": GoogleSearch,
        "text_to_speech": TTS,
        "image_caption": ImageCaption,
        "text_to_image": TextToImage,
        "text_to_video": TextToVideo,
        "image_to_prompt": ImageToPrompt,
        "search_doc": SearchDoc,
        "bash_shell": RunShell,
        "get_future_weather": GetFutureWeather,
        "get_today_weather": GetTodayWeather,
        "wikipedia": Wikipedia,
        "web_page": WebPage,
        "wolfram_alpha": WolframAlpha,
        "duckduckgo": DuckDuckGo,
        "search_author_by_name": SearchAuthorByName,
        "search_author_by_interests": SearchAuthorByInterests,
        "author_uid2paper": AuthorUID2Paper,
        "search_paper": SearchPaper,
        "search_single_paper": SearchSinglePaper,
        "search_related_paper": SearchRelatedPaper,
        "search_cite_paper": SearchCitePaper,
        "search_paper_by_keywords": SearchPaperByKeywords,
    }
    if name not in name2tool:
        raise NotImplementedError
    return name2tool[name]
```