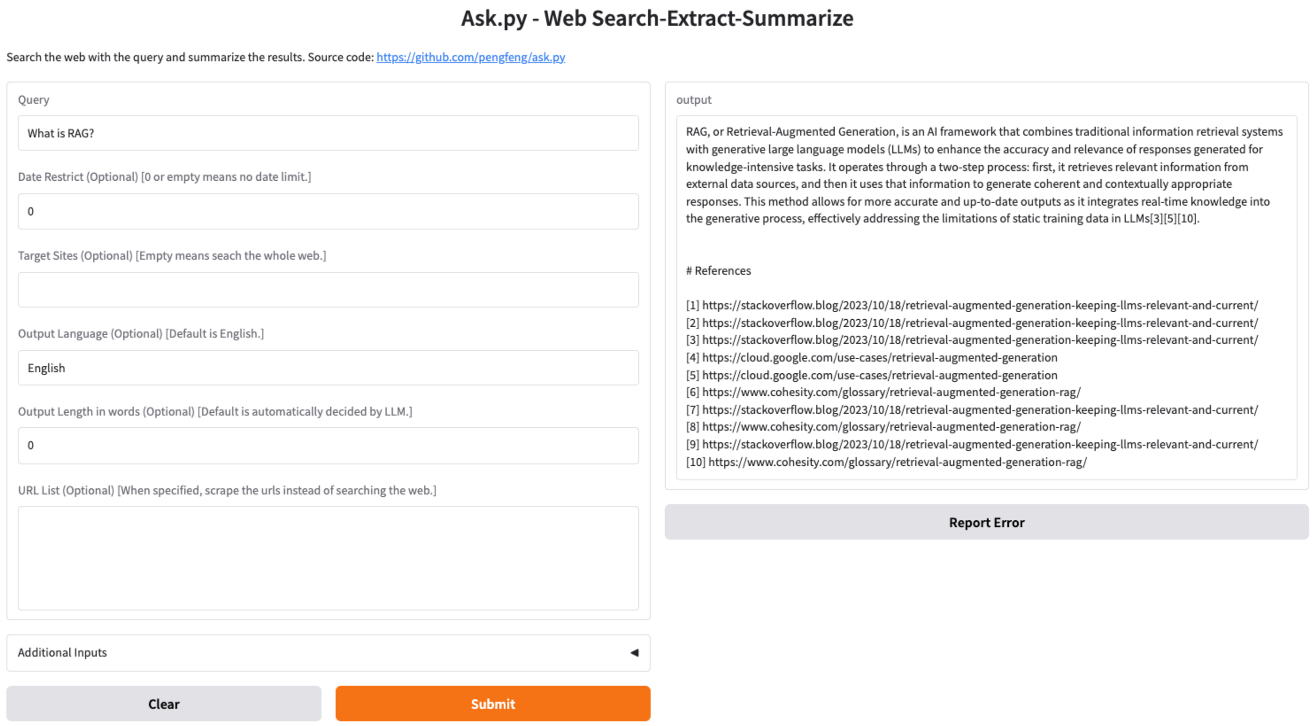
ask.py

license MIT

A single Python program to implement the search-extract-summarize flow, similar to AI search engines such as Perplexity.

- You can run it on command line or with a GradIO UI.
- You can control the output behavior, e.g., extract structured data or change output language,
- You can control the search behavior, e.g., restrict to a specific site or date, or just scrape a specified list of URLs.
- You can run it in a cron job or bash script to automate complex search/data extraction tasks.

We have a running UI example in HuggingFace Spaces.



Ask.py - Web Search-Extract-Summarize

Search the web with the query and summarize the results. Source code: https://github.com/pengfeng/ask.py

Query
What is RAG?

Date Restrict (Optional) [0 or empty means no date limit.]
0

Target Sites (Optional) [Empty means seach the whole web.]

Output Language (Optional) [Default is English.]
English

Output Length in words (Optional) [Default is automatically decided by LLM.]
0

URL List (Optional) [When specified, scrape the urls instead of searching the web.]

output

```
RAG, or Retrieval-Augmented Generation, is an AI framework that combines traditional information retrieval systems
with generative large language models (LLMs) to enhance the accuracy and relevance of responses generated for
knowledge-intensive tasks. It operates through a two-step process: first, it retrieves relevant information from
external data sources, and then it uses that information to generate coherent and contextually appropriate
responses. This method allows for more accurate and up-to-date outputs as it integrates real-time knowledge into
the generative process, effectively addressing the limitations of static training data in LLMs[3][5][10].


# References

[1] https://stackoverflow.blog/2023/10/18/retrieval-augmented-generation-keeping-llms-relevant-and-current/
[2] https://stackoverflow.blog/2023/10/18/retrieval-augmented-generation-keeping-llms-relevant-and-current/
[3] https://stackoverflow.blog/2023/10/18/retrieval-augmented-generation-keeping-llms-relevant-and-current/
[4] https://cloud.google.com/use-cases/retrieval-augmented-generation
[5] https://cloud.google.com/use-cases/retrieval-augmented-generation
[6] https://www.cohesity.com/glossary/retrieval-augmented-generation-rag/
[7] https://stackoverflow.blog/2023/10/18/retrieval-augmented-generation-keeping-llms-relevant-and-current/
[8] https://www.cohesity.com/glossary/retrieval-augmented-generation-rag/
[9] https://stackoverflow.blog/2023/10/18/retrieval-augmented-generation-keeping-llms-relevant-and-current/
[10] https://www.cohesity.com/glossary/retrieval-augmented-generation-rag/
```

> [!NOTE]
>
> - Our main goal is to illustrate the basic concepts of AI search engines with the raw constructs.
>   Performance or scalability is not in the scope of this program.
> - We are planning to open source a real search-enabled AI toolset with real DB setup, real document
>   pipeline, and real query engine soon. Star and watch this repo for updates!

> [UPDATE]

> - 2024-11-10: add Chonkie as the default chunker
> - 2024-10-28: add extract function as a new output mode
> - 2024-10-25: add hybrid search demo using DuckDB full-text search
> - 2024-10-22: add GradIO integation
> - 2024-10-21: use DuckDB for the vector search and use API for embedding
> - 2024-10-20: allow to specify a list of input urls
> - 2024-10-18: output-language and output-length parameters for LLM
> - 2024-10-18: date-restrict and target-site parameters for seach

The search-extract-summarize flow

Given a query, the program will

- search Google for the top 10 web pages
- crawl and scape the pages for their text content
- chunk the text content into chunks and save them into a vectordb
- perform a vector search with the query and find the top 10 matched chunks
- [Optional] search using full-text search and combine the results with the vector search
- [Optional] use a reranker to re-rank the top chunks
- use the top chunks as the context to ask an LLM to generate the answer
- output the answer with the references

Of course this flow is a very simplified version of the real AI search engines, but it is a good starting point to understand the basic concepts.

One benefit is that we can manipulate the search function and output format.

For example, we can:

- search with date-restrict to only retrieve the latest information.
- search within a target-site to only create the answer from the contents from it.
- ask LLM to use a specific language to answer the question.
- ask LLM to answer with a specific length.
- crawl a specific list of urls and answer based on those contents only.

Quick start

```
# recommend to use Python 3.10 or later and use venv or conda to create a virtual environment
% pip install -r requirements.txt

# modify .env file to set the API keys or export them as environment variables as below

# right now we use Google search API
% export SEARCH_API_KEY="your-google-search-api-key"
% export SEARCH_PROJECT_KEY="your-google-cx-key"

# right now we use OpenAI API
% export LLM_API_KEY="your-openai-api-key"

# By default, the program will start a web UI. See GradIO Deployment section for more info.
# Run the program on command line with -c option
% python ask.py -c -q "What is an LLM agent?"

# we can specify more parameters to control the behavior such as date_restrict and target_site
% python ask.py --help
Usage: ask.py [OPTIONS]

  Search web for the query and summarize the results.

Options:
  -q, --query TEXT                Query to search
  -o, --output-mode [answer|extract]
                                  Output mode for the answer, default is a
                                  simple answer
  -d, --date-restrict INTEGER     Restrict search results to a specific date
                                  range, default is no restriction
  -s, --target-site TEXT          Restrict search results to a specific site,
                                  default is no restriction
  --output-language TEXT          Output language for the answer
  --output-length INTEGER         Output length for the answer
  --url-list-file TEXT            Instead of doing web search, scrape the
                                  target URL list and answer the query based
                                  on the content
  --extract-schema-file TEXT      Pydantic schema for the extract mode
  --inference-model-name TEXT     Model name to use for inference
  --hybrid-search                 Use hybrid search mode with both vector
                                  search and full-text search
  -c, --run-cli                   Run as a command line tool instead of
                                  launching the Gradio UI
  -l, --log-level [DEBUG|INFO|WARNING|ERROR]
                                  Set the logging level  [default: INFO]
  --help                          Show this message and exit.
```

Libraries and APIs used

- Google Search API
- OpenAI API

- Jinja2
- bs4
- DuckDB
- GradIO
- Chonkie

GradIO Deployment

> [!NOTE]
> Original GradIO app-sharing document here.

Quick test and sharing

By default, the program will start a web UI and share through GradIO.

```
% python ask.py
* Running on local URL:  http://127.0.0.1:7860
* Running on public URL: https://77c277af0330326587.gradio.live

# you can also specify SHARE_GRADIO_UI to only run locally
% export SHARE_GRADIO_UI=False
% python ask.py
* Running on local URL:  http://127.0.0.1:7860
```

To share a more permanent link using HuggingFace Spaces

- First, you need to create a free HuggingFace account.
- Then in your settings/token page, create a new token with Write permissions.
- In your terminal, run the following commands in you app directory to deploy your program to HuggingFace Spaces:

```
% pip install gradio
% gradio deploy
Creating new Spaces Repo in '/home/you/ask.py'. Collecting metadata, press Enter to accept default value.
Enter Spaces app title [ask.py]: ask.py
Enter Gradio app file [ask.py]:
Enter Spaces hardware (cpu-basic, cpu-upgrade, t4-small, t4-medium, l4x1, l4x4, zero-a10g, a10g-small, a10g-large, a10g-largex2, a10g-largex4, a100-large, v5e-1x1, v5e-2x2, v5e-2x4) [cpu-basic]:
Any Spaces secrets (y/n) [n]: y
Enter secret name (leave blank to end): SEARCH_API_KEY
Enter secret value for SEARCH_API_KEY: YOUR_SEARCH_API_KEY
Enter secret name (leave blank to end): SEARCH_PROJECT_KEY
Enter secret value for SEARCH_API_KEY: YOUR_SEARCH_PROJECT_KEY
Enter secret name (leave blank to end): LLM_API_KEY
Enter secret value for LLM_API_KEY: YOUR_LLM_API_KEY
Enter secret name (leave blank to end):
Create Github Action to automatically update Space on 'git push'? [n]: n
Space available at https://huggingface.co/spaces/your_user_name/ask.py
```

Now you can use the HuggingFace space app to run your queries.

Use Cases

- Search like Perplexity
- Only use the latest information from a specific site
- Extract information from web search results