Rufus Documentation

Rufus is a powerful web crawling and content extraction library designed for RAG pipelines. It combines intelligent web crawling with semantic content extraction to gather relevant information based on specific user instructions.

Overview

Rufus consists of three main components:

1. WebCrawler: Recursively crawls websites to gather content
2. ContentExtractor: Processes and extracts relevant information using semantic similarity
3. RufusClient: Main interface that orchestrates crawling and extraction

Installation and Running

```
*git clone https://github.com/Aadyant12/Chima-Rufus-AI-Agent.git*

*# Required dependencies*
*pip install beautifulsoup4 requests sentence-transformers torch*

*from client import RufusClient*

*client = RufusClient(api_key="loved_the_assignment")*

*documents = client.scrape(*
   *url="https://www.withchima.com/",*
   *instructions="Find information about product features and pricing",*
   *max_depth=2,*
*)*

*documents*
```

WebCrawler

The WebCrawler component handles recursive website crawling with the following features:

1. Depth-limited crawling
2. Rate limiting
3. URL filtering
4. Robust error handling

```*python*
*from rufus import WebCrawler*

*crawler = WebCrawler()*
```

```
pages = crawler.crawl(
    start_url="https://example.com",
    max_depth=3
)
```

ContentExtractor

The ContentExtractor uses semantic similarity to identify and extract relevant content:

1. Semantic similarity matching using Sentence Transformers
2. Automatic content chunking
3. Text summarization: The content extractor returns the complete relevant content (which can be further embedded for RAG usage) as well as a summary of the content (for transparency and human understanding)
4. Relevance scoring

```python
from rufus import ContentExtractor

extractor = ContentExtractor()
content = extractor.extract(
    pages=crawled_pages,
    instructions="Extract information about machine learning applications"
)
```

RufusClient

The RufusClient provides a simplified interface to the entire pipeline:

```python
from rufus import RufusClient

client = RufusClient(api_key="loved_the_assignment")
results = client.scrape(
    url="https://example.com",
    instructions="Find information about deep learning architectures",
    max_depth=3
)
```

RAG Pipeline Integration

After retrieval of relevant content from different websites, the contents outputted in the json file by Rufus can be embedded and stored in vector databases like Faiss or Annoy. The vector DBs can then be used in conjunction with an LLM to have an advanced RAG system.

Advanced Configuration

Customizing the WebCrawler:

1. The delay request can be modified
2. A custom URL filtering can incorporated by modifying the *_should_crawl* method

Customizing the ContentExtractor:

1. A different embedding model can be used for enhanced similarity search
2. Chunk sizes can be modified

Error Handling

Rufus provides a custom `RufusError` exception class for error handling. Common errors to handle:

1. Invalid API key
2. Network connectivity issues
3. Rate limiting
4. Invalid URLs
5. Parsing errors

Performance Tips

1. Use appropriate chunk sizes based on your content
2. Implement caching for frequently accessed pages
3. Use batch processing for large-scale crawling
4. Monitor and adjust similarity thresholds
5. Implement proper cleanup of resources

Challenges faced

1. Memory management: Large websites crashed the crawler. Solved using generator patterns and page depth limit.
2. Performance: Semantic similarity was slow. Added batch processing and chunking.
3. Content quality - Initial results were noisy. Implemented text filters for length and boilerplate content.
4. Error handling - Network issues caused crashes. Added retries, timeouts and better exception handling.