

Deep research

RAG技术演进三部曲：从信息拼接走向认知革命

在**Simple RAG时代**，检索→增强→生成的三段式管道，各模块相互独立，每个环节独立完成。这种方法虽然让我们在信息检索和生成上取得了一些进展，但始终存在信息割裂和效率低下的问题。

到了**Graph RAG时代**，知识图谱赋予RAG认知骨架，检索和带有推理能力的内容生成开始变得可能，一方面可以通过实体关系链实现多跳问答，另一方面通过实时图谱更新应对动态变化的信息。虽然如此，如何深度拆解问题、精准地检索相关信息，仍然处于初步的探索阶段。问题的拆解和内容的生成依然离不开大量的人工干预。

然而，进入了**Deep Research时代**，认知智能的爆发，一切变得更为顺畅。从问题的提问到拆解，再到针对不同维度的内容检索、最后的生成，所有环节已经能够在统一的流程中执行，RAG不再只是一个工具，而是成为了一个完整的问题解决者，例如，可以自主拆解模糊需求（如将"帮我做市场分析"分解为行业趋势、竞品动态、用户洞察等12个子任务）。这种进化不仅改变了我们处理问题的方式，也为各种复杂任务提供了更高效的解决方案。

DeepSearcher核心思路

在OpenAI发布相关功能后，许多开源贡献者积极进行分析与复现，目前Github上已有多种开源方案，基本实现思路如下：

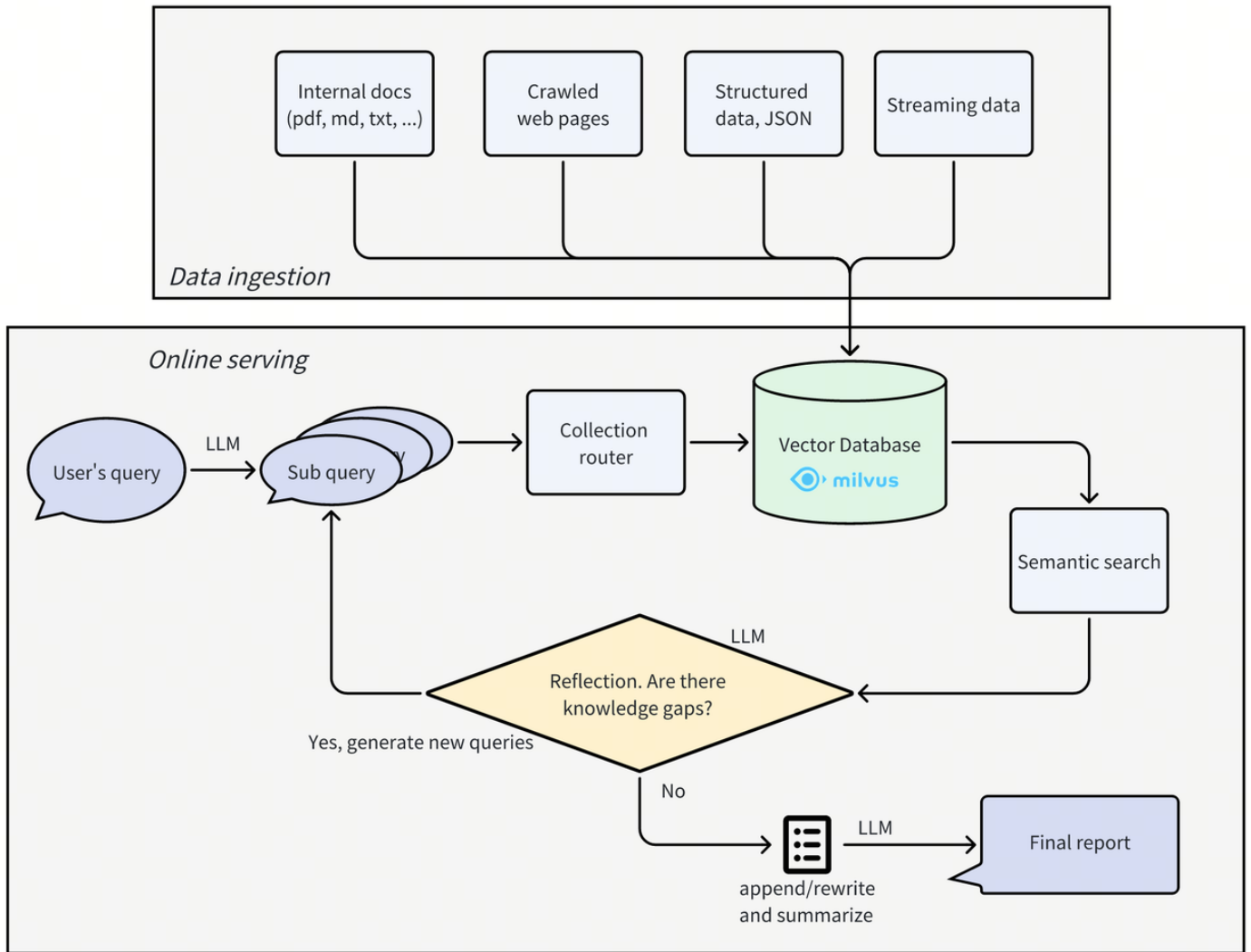
1. **问题分析**：大模型分析用户输入，确定回答问题所需的角度与步骤。许多大模型（如DeepSeek、ChatGPT、Gemini等）通过勾选推理选项即可自动生成这一过程。
2. **在线搜索**：基于生成的问题，大模型执行在线搜索，获取前k项相关结果，并将其返回给模型。
3. **内容总结**：大模型对搜索结果进行总结，提炼出简洁的答案。
4. **答案判定**：大模型综合所有信息，判断答案是否完整和准确。若答案符合要求，直接输出。如果达到设定的循环次数或token上限，仍然输出当前答案；否则，生成新的问题，重新进入第一步，并带入历史信息进行更新。

Deep Searcher的实现思路

<https://github.com/zilliztech/deep-searcher>

Deep Searcher基于大部分开源Deep Research方案基础上进行改良，其方案架构如下图所示。

DeepSearcher



Deep Searcher通过引入向量数据库Milvus,可以对用户存储在本地的数据进行海量低延时的离线搜索。

其实现流程包括以下三步：

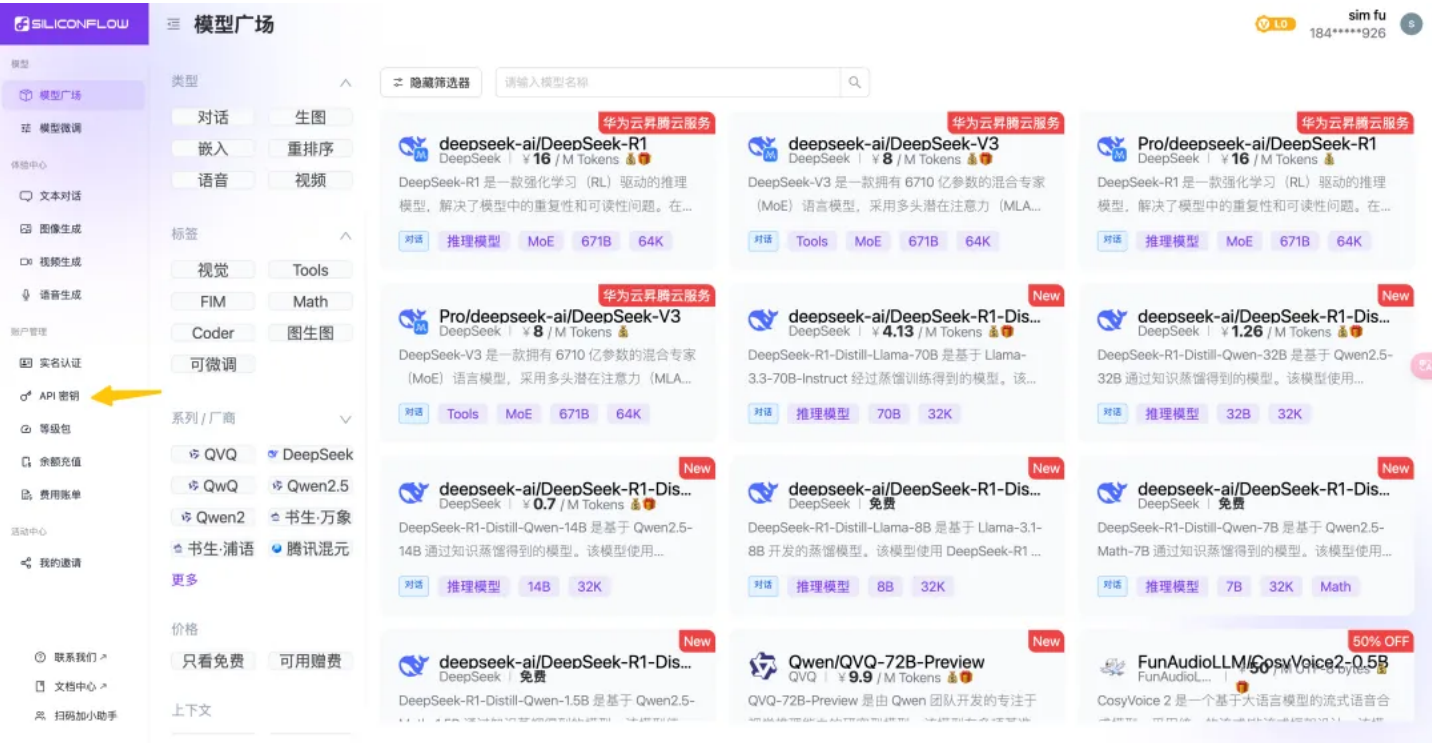
- 1. 问题分析：**接收用户问题后，LLM分析并生成多个子问题，同时确定每个子问题所需的数据集合。
- 2. 信息检索：**根据分析结果，从向量数据库中检索相关信息。需要注意，向量数据库中的数据为离线数据，因此需提前将内部数据、在线下载数据或流系统定期输入的数据写入数据库。
- 3. 内容判定：**检索到相关信息后，将用户问题、子问题及其搜索结果交由大模型进行判断。如果问题已得到完整解答，则输出最终答案；如果达到设定的循环次数或token上限，则结束并输出当前答案。否则，大模型会生成新的问题，进入下一轮循环。

Deep Searcher私有化部署实践

下面以DeepSeek+DeepSearcher+硅基流动为例实现Deep Searcher的私有化部署。

第一步：硅基流动准备工作

- 1.注册硅基流动账号，访问 [siliconflow](#) 官网进行注册
- 2.创建API Key，完成后需要保存个人的API Key，文章后续需要使用该Key，另外个人Key注意保密，如果出现泄漏可以删除重新生成。



第二步：DeepSearcher运行环境准备

- 1.从Github上获取项目源码

代码块

```
1 git clone https://github.com/zilliztech/deep-searcher.git
```

- 2.为DeepSearcher创建虚拟python环境，建议使用python3.10以上版本，下面使用python自带的venv创建虚拟环境，也可以根据自己熟悉的工具进行创建。

代码块

```
1 cd deep-searcherpython3 -m venv .venvsourcesource .venv/bin/activate
```

- 3.安装deepsearcher及其依赖第三方库

代码块

```
1 pip install -e .
```

第三步：DeepSearcher 示例运行

1.设置环境变量，以下以Linux为例，编辑用户目录下的.bashrc，添加SILICONFLOW_API_KEY环境变量，其值设置为自己的API Key。

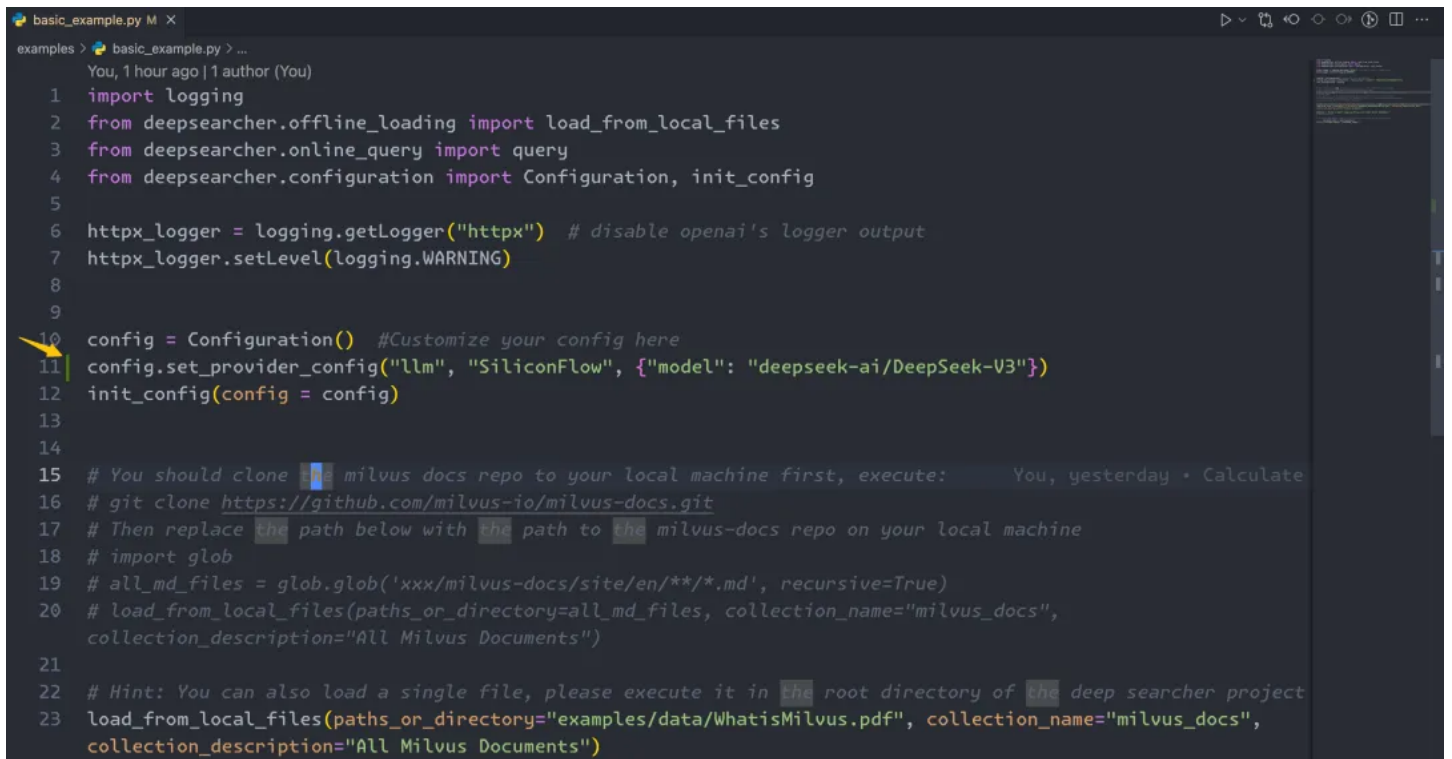
代码块

```
1 export SILICONFLOW_API_KEY=sk-xxx
```

2.编辑项目DeepSearcher项目下的examples目录下的basic_example.py文件，设置硅基流动的DeepSeek服务。下面示例中设置的是deep seek v3的模型，也可以设置为r1。（即：deepseek-ai/DeepSeek-R1）

代码块

```
1 config.set_provider_config("llm", "SiliconFlow", {"model": "deepseek-ai/DeepSeek-V3"})
```



```
basic_example.py M X
examples > basic_example.py > ...
You, 1 hour ago | 1 author (You)
1 import logging
2 from deepsearcher.offline_loading import load_from_local_files
3 from deepsearcher.online_query import query
4 from deepsearcher.configuration import Configuration, init_config
5
6 httpx_logger = logging.getLogger("httpx") # disable openai's logger output
7 httpx_logger.setLevel(logging.WARNING)
8
9
10 config = Configuration() #Customize your config here
11 config.set_provider_config("llm", "SiliconFlow", {"model": "deepseek-ai/DeepSeek-V3"})
12 init_config(config = config)
13
14
15 # You should clone the milvus docs repo to your local machine first, execute: You, yesterday • Calculate
16 # git clone https://github.com/milvus-io/milvus-docs.git
17 # Then replace the path below with the path to the milvus-docs repo on your local machine
18 # import glob
19 # all_md_files = glob.glob('xxx/milvus-docs/site/en/**/*.md', recursive=True)
20 # load_from_local_files(paths_or_directory=all_md_files, collection_name="milvus_docs",
21 # collection_description="All Milvus Documents")
22
23 # Hint: You can also load a single file, please execute it in the root directory of the deep searcher project
24 load_from_local_files(paths_or_directory="examples/data/WhatIsMilvus.pdf", collection_name="milvus_docs",
25 collection_description="All Milvus Documents")
```

3.在运行前，这里简单介绍下这个示例，是加载一个本地的pdf，然后让大模型针对于这个pdf回答问题，当然你也可以通过同样的方式load更多pdf。

代码块

```
1 # 确保你在该项目的根目录
2 python examples/basic_example.py
```

运行后，将会看到整个问题思考解答的部分流程，在最后会给出一个类似于下图的最终答案。

```
==== FINAL ANSWER====

### Report: Comparison of Milvus with Other Vector Databases

#### Overview
Milvus is a high-performance, highly scalable vector database designed to handle unstructured data such as text, images, and audio. It is available as both open-source software and a cloud service, and it supports a wide range of data types and deployment modes, from local prototyping to large-scale distributed systems.

#### Key Features of Milvus
1. **Advanced Search Algorithms**: Milvus supports optimized in-memory and on-disk indexing/search algorithms like IVF, HNSW, and DiskANN, delivering 30%-70% better performance compared to FAISS and HNSWLib.
2. **High-Performance Search Engine**: Built in C++, Milvus's search engine integrates hardware-aware code optimizations, including assembly-level vectorization and multi-thread parallelization.
3. **Column-Oriented Architecture**: This design reduces data access and enhances performance by reading only specific fields involved in queries and enabling vectorized operations on entire columns.
4. **Scalability**: Milvus can handle tens of billions of vectors, supported by its cloud-native, decoupled system architecture, which allows for easy scaling with Kubernetes or public clouds.

#### Main Use Cases and Applications
Milvus is used for applications requiring fast and scalable searches and analytics on unstructured data, such as recommendation systems, image and video search, and natural language processing. It powers large-scale scenarios for over 300 major enterprises, including Salesforce, PayPal, and Airbnb.

#### Performance Comparison: Scalability, Speed, and Accuracy
- **Scalability**: Milvus supports tens of billions of vectors, making it highly scalable for enterprise-level applications. Its cloud-native architecture ensures continuous expansion as data grows.
- **Speed**: Milvus achieves high performance through advanced search algorithms, a C++ search engine, and column-oriented data access patterns.
- **Accuracy**: Optimized search algorithms and hardware-aware optimizations ensure high accuracy in vector searches.

#### Advantages and Disadvantages
- **Advantages**:
  - Superior performance with optimized search algorithms.
  - High scalability and stability for large-scale applications.
  - Flexibility in deployment modes, from local prototyping to distributed systems.
  - Broad support for data types and robust data modeling capabilities.
- **Disadvantages**:
  - Complexity in setup and management in highly distributed environments.
  - Requires expertise in hardware-aware optimizations to fully leverage its capabilities.

#### Conclusion
Milvus stands out as a leading vector database due to its high performance, scalability, and flexibility. It is particularly well-suited for enterprise applications requiring large-scale, high-speed searches and analytics on unstructured data. While it may present some complexity in setup and management, its robust features and optimizations make it a strong contender in the vector database landscape.
Consumed tokens: 12224
```

这个示例最后也打印出了消耗的token数目，同时也可以到硅基流动平台查看该次消耗的费用。运行这次示例，我消耗了12224 token数目，费用为0.0286元。

SILICONFLOW

费用账单

sim fu 184*****926

模型

模型广场

模型微调

体验中心

文本对话

图像生成

视频生成

语音生成

账户管理

实名认证

API 密钥

等级包

余额充值

费用账单

联系我们

文档中心

扫码加小助手

月度账单

费用明细

2025-02-18

API 密钥: Select API Key

产品类型	产品名称	用量	消费总额 (元)	消费时间段
对话模型	deepseek-ai/DeepSeek-V3	12224 Token(s)	0.0286	22:00-22:59

< 1 >