多跳推理知识问答系统

这是一个基于RAG (Retrieval-Augmented Generation) 技术的知识问答系统，支持多知识库管理、多轮对话和创新的多跳推理功能。系统可以通过本地知识库和联网搜索获取信息，为电子制造业相关问题提供智能回答。

1. 文件结构

```
.
├── rag.py                # 主程序，包含RAG系统的核心功能实现
├── config.py             # 配置文件，包含各种参数设置
├── text2vec.py           # 文本向量化工具
├── retrievor.py          # 联网搜索和文本检索功能
├── knowledge_bases/      # 知识库根目录
│   └── default/          # 默认知识库
├── output_files/         # 临时输出文件目录
```

2. 安装与部署

环境要求

- Python 3.10+
- CUDA 支持（可选，用于GPU加速）

依赖库

```
pip install -r requirements.txt
```

快速部署

1. 配置API密钥

编辑 `config.py` 文件，设置必要的API密钥：

```
# 向量化API配置（阿里云通义千问）
api_key = "sk-你的API密钥"
base_url = "https://dashscope.aliyuncs.com/compatible-mode/v1"

# LLM API配置（阿里云通义千问）
llm_api_key = "sk-你的API密钥"
llm_base_url = "https://dashscope.aliyuncs.com/compatible-mode/v1"
llm_model = "qwen-plus"
```

配置API密钥（新用户免费赠送100万token）

https://bailian.console.aliyun.com/?spm=a2c4g.11186623.0.0.3c4a72a3Z9MBH1#/home

2. 启动系统

```
python rag.py
```

系统将在以下地址启动：

- 本地访问：http://localhost:7860
- 公网访问：启动时会显示share链接

3. 系统功能

- 📚 **多知识库管理**：创建、删除、查看多个知识库
- 📄 **文件导入**：支持上传TXT和PDF文件到知识库
- 🔍 **语义检索**：基于向量相似度的文档检索
- 🔄 **多跳推理**：创新的多跳推理机制，通过迭代检索和推理找到更全面的答案
- 🌐 **联网搜索**：集成web搜索功能补充知识库信息
- 💬 **多轮对话**：支持基于历史上下文的对话
- 🚀 **流式响应**：实时显示检索和推理过程
- 📊 **表格输出**：支持以Markdown表格输出结构化信息

4. 核心算法

4.1 语义分块 (Semantic Chunking)

系统使用增强的句子分割器将文档分成语义连贯的块，优化检索效果：

```
def semantic_chunk(text: str, chunk_size=800, chunk_overlap=20) -> List[dict]:
    class EnhancedSentenceSplitter(SentenceSplitter):
        # 增强的分句器，支持中文标点
        def __init__(self, *args, **kwargs):
            custom_seps = ["; ", "!", "?", "\n"]
            separators = [kwargs.get("separator", "。")] + custom_seps
            kwargs["separator"] = '|'.join(map(re.escape, separators))
            super().__init__(*args, **kwargs)
```

4.2 多跳推理RAG系统

多跳推理是系统的核心创新点，实现了迭代式的检索和推理过程：

```
class ReasoningRAG:
    """
    多跳推理RAG系统，通过迭代式的检索和推理过程回答问题，支持流式响应
    """
```

```
    def retrieve_and_answer(self, query: str, use_table_format: bool = False) ->
Tuple[str, Dict[str, Any]]:
        """
        执行多跳检索和回答生成的主要方法
        """
        # 初始检索
        query_vector = self._vectorize_query(query)
        initial_chunks = self._retrieve(query_vector, self.initial_candidates)

        # 初始推理
        reasoning = self._generate_reasoning(query, initial_chunks, hop_number=0)

        # 迭代跳数进行检索和推理
        hop = 1
        while (hop < self.max_hops and
               not reasoning["is_sufficient"] and
               reasoning["follow_up_queries"]):

            # 处理每个后续查询
            for follow_up_query in reasoning["follow_up_queries"]:
                # 为后续查询检索
                follow_up_vector = self._vectorize_query(follow_up_query)
                follow_up_chunks = self._retrieve(follow_up_vector,
self.refined_candidates)

            # 为此跳数生成推理
            reasoning = self._generate_reasoning(
                query,
                hop_chunks,
                previous_queries=all_queries[:-1],
                hop_number=hop
            )

            hop += 1

        # 合成最终答案
        answer = self._synthesize_answer(query, all_chunks, reasoning_steps,
use_table_format)
```

4.3 向量化与检索

系统使用FAISS进行高效的向量检索：

```
def build_faiss_index(vector_file, index_path, metadata_path):
    # 从向量数据构建FAISS索引
    vectors = [item['vector'] for item in valid_data]
    vectors = np.array(vectors, dtype=np.float32)

    # 根据向量数量选择索引类型
    if nlist >= 1 and n_vectors >= nlist * 39:
        quantizer = faiss.IndexFlatIP(dim)
```

```
            index = faiss.IndexIVFFlat(quantizer, dim, nlist)
            if not index.is_trained:
                index.train(vectors)
            index.add(vectors)
        else:
            index = faiss.IndexFlatIP(dim)
            index.add(vectors)
```

5. API集成

系统支持通过OpenAI兼容接口调用第三方嵌入和LLM服务：

```
def vectorize_query(query, model_name=Config.model_name, batch_size=Config.batch_size) ->
np.ndarray:
    """向量化文本查询，返回嵌入向量"""
    embedding_client = OpenAI(
        api_key=Config.api_key,
        base_url=Config.base_url
    )

    completion = embedding_client.embeddings.create(
        model=model_name,
        input=batch,
        dimensions=Config.dimensions,
        encoding_format="float"
    )
```

6. 使用指南

6.1 知识库管理

1. 创建知识库
    - 在"知识库管理"标签页输入知识库名称
    - 点击"创建知识库"按钮
2. 上传文件
    - 选择要上传的TXT或PDF文件
    - 点击上传按钮
    - 系统会自动处理文件并构建索引

6.2 提问方式

1. 切换到"对话交互"标签页
2. 选择要使用的知识库
3. 配置对话设置：
    - 启用/禁用联网搜索

    - 启用/禁用表格格式输出
    - 启用/禁用多跳推理
4. 输入问题并提交
5. 查看检索进展和回答结果