

Dify 知识库召回率评测完整指南

从零开始搭建 RAG 检索效果评估系统

RAG 评测技术文档

2026 年 2 月 4 日

目录

1 概述与背景	3
1.1 什么是 RAG 召回率评测	3
1.2 为什么需要系统化的评测	3
1.3 核心评测指标详解	4
2 环境准备	4
2.1 系统要求	4
2.2 Python 环境安装	5
2.2.1 Windows 系统安装步骤	5
2.2.2 macOS 系统安装步骤	5
2.2.3 Linux 系统安装步骤	6
2.3 创建项目目录	6
2.4 创建 Python 虚拟环境	6
2.5 安装依赖库	7
2.6 获取 Dify API 配置	8
2.6.1 获取 API Key	8
2.6.2 获取知识库 ID	8
2.6.3 配置环境变量文件	8
3 评测集构建	9
3.1 评测集的重要性	9
3.2 评测集数据格式	10
3.3 评测集示例	10
3.4 评测集构建工具	11

3.5 脚本使用方法	17
3.5.1 方式一：创建空白模板	17
3.5.2 方式二：从知识库自动提取候选问题	17
3.6 人工审核与标注	17
4 执行检索评测	18
4.1 核心评测模块	18
4.2 运行评测	29
4.2.1 基础评测命令	29
4.2.2 启用重排器的评测	30
5 批量对比评测	30
5.1 批量评测脚本	30
6 可视化与报告生成	34
6.1 可视化脚本代码	34
6.2 运行可视化脚本	45
7 一键执行完整流程	45
7.1 整合脚本	46
7.2 快速上手指南	48
8 常见问题与解决方案	49
8.1 API 调用错误	49
8.2 评测结果异常	50
8.3 图表中文显示异常	50
9 附录	50
9.1 项目目录结构	50
9.2 评测指标数学定义	51
9.3 参考资料	52

1 概述与背景

1.1 什么是 RAG 召回率评测

当你使用 Dify 搭建知识库问答系统时，整个问答流程可以分为两个阶段：第一阶段是“检索”，系统根据用户的问题，从知识库中找出最相关的文档片段；第二阶段是“生成”，大语言模型基于检索到的内容，生成最终的回答。

召回率评测关注的是第一阶段——检索环节的效果。具体来说，它要回答一个核心问题：当用户提出问题时，系统能否准确地从知识库中找到包含正确答案的文档或段落？

这个问题非常关键。因为如果检索阶段就没有找到正确的内容，那么无论大模型多么强大，也无法给出准确的回答。可以说，检索质量是整个 RAG 系统的基础。

1.2 为什么需要系统化的评测

很多人在搭建知识库问答系统时，会采用“手工测试”的方式来评估效果：随机问几个问题，看看回答得对不对。这种方式存在几个明显的问题：

首先，样本量太小，不具有统计意义。你测试的那几个问题可能恰好都能回答正确，但换一批问题可能就不行了。

其次，缺乏可比性。当你调整了分块策略或修改了参数后，很难客观判断效果是变好了还是变差了。

最后，无法持续监控。知识库会不断更新，新增文档、修改内容，如果没有系统化的评测，你无法知道这些变化对检索效果的影响。

系统化的召回率评测可以解决以上所有问题。通过构建标准化的评测集，使用统一的评测指标，你可以：

- **客观对比不同配置的效果：** Dify 提供了通用分块、父子分块、QA 分块三种索引方式，每种方式还有不同的参数可以调整。通过评测，你可以用数据说明哪种配置最适合你的场景。
- **科学地进行参数调优：** chunk_size 设多大？overlap 要不要？TopK 取几最合适？这些问题都可以通过评测来回答，而不是凭感觉猜测。
- **建立效果基线，持续监控：** 每次知识库更新后，跑一遍评测，就能知道效果有没有下降。如果发现问题，可以及时排查原因。
- **评估新技术的收益：** 比如是否要引入重排器（Reranker），能带来多少提升？值不值得为此增加延迟和成本？这些决策都需要数据支撑。

1.3 核心评测指标详解

在正式开始评测之前，我们需要先了解几个核心指标。这些指标是评估检索效果的标准语言，理解它们的含义非常重要。

指标	计算公式	含义说明
Recall@K	$\frac{\text{命中的问题数}}{\text{总问题数}}$	在所有测试问题中，有多少问题能在 Top K 个检索结果里找到正确答案
Precision@K	$\frac{\text{命中数}}{K \times \text{总问题数}}$	检索返回的结果中，有多少是真正相关的
F1@K	$\frac{2 \times P \times R}{P + R}$	召回率和精确率的调和平均，综合衡量检索效果
MRR	$\frac{1}{N} \sum_{i=1}^N \frac{1}{rank_i}$	正确答案首次出现位置的倒数平均值，反映排序质量

表 1: 核心评测指标说明

为了更好地理解这些指标，我们来看一个具体的例子：

假设你有一个问题“年假有多少天”，系统返回了 5 条检索结果（Top5）。经过核对，第 2 条结果中包含了正确答案“员工入职满一年，可享受 5 天带薪年假”。

在这个例子中：

- 这个问题“命中”了，因为 Top5 中有正确答案
- 命中排名是 2，因为正确答案出现在第 2 条
- 这个问题对 MRR 的贡献是 $\frac{1}{2} = 0.5$

如果我们测试了 100 个问题，其中 80 个问题在 Top5 中找到了正确答案，那么 $Recall@5 = 80\%$ 。

2 环境准备

本节将详细介绍运行评测脚本所需的环境配置。即使你之前没有 Python 开发经验，按照以下步骤操作也能顺利完成配置。

2.1 系统要求

在开始之前，请确认你的电脑满足以下基本要求：

- 操作系统：Windows 10/11、macOS 10.15 及以上、Ubuntu 18.04 及以上版本均可

- **Python 版本**: 需要 Python 3.8 或更高版本, 推荐使用 Python 3.10 或 3.11
- **内存**: 建议 8GB 以上, 如果评测集较大或需要生成复杂图表, 16GB 会更流畅
- **网络**: 需要能够访问你的 Dify 服务, 无论是本地部署还是云端版本

2.2 Python 环境安装

2.2.1 Windows 系统安装步骤

Windows 用户请按照以下步骤安装 Python:

1. 打开浏览器, 访问 Python 官方网站: <https://www.python.org/downloads/>
2. 在页面上找到 Python 3.10 或 3.11 版本的下载链接, 点击下载。注意: 不建议下载最新版本, 因为部分依赖库可能还未完全适配。
3. 下载完成后, 双击安装程序。在安装界面的第一页, **务必勾选底部的”Add Python to PATH” 选项**。这一步非常关键, 如果忘记勾选, 后续在命令行中将无法直接使用 python 命令。
4. 点击”Install Now” 开始安装, 等待安装完成。
5. 安装完成后, 验证是否安装成功。按下 Win+R 键打开运行对话框, 输入 cmd 并回车, 打开命令提示符窗口, 输入以下命令:

```
1 python --version
2 # 如果安装成功, 应该显示类似: Python 3.10.11 或 Python 3.11.4
```

2.2.2 macOS 系统安装步骤

macOS 用户有两种安装方式可以选择。

方式一：使用 Homebrew 安装（推荐）

如果你的 Mac 上已经安装了 Homebrew (macOS 上最流行的包管理器), 可以通过一行命令完成安装:

```
1 # 使用 Homebrew 安装 Python 3.11
2 brew install python@3.11
3
4 # 安装完成后, 验证版本
5 python3 --version
```

方式二：从官网下载安装包

如果没有安装 Homebrew, 可以直接从官网下载安装包。访问 <https://www.python.org/downloads/macos/>, 下载 Python 3.10 或 3.11 的 macOS 安装包, 双击运行并按提示完成安装。

2.2.3 Linux 系统安装步骤

大多数 Linux 发行版都预装了 Python，但版本可能较旧。以 Ubuntu/Debian 系统为例，可以通过以下命令安装指定版本：

```
1 # 更新软件包列表
2 sudo apt update
3
4 # 安装 Python 3.10 及相关工具
5 sudo apt install python3.10 python3.10-venv python3-pip
6
7 # 验证安装
8 python3 --version
```

2.3 创建项目目录

选择一个合适的位置创建项目目录。后续所有的代码文件、配置文件和输出结果都将存放在这个目录中。

```
1 # Windows 用户 (在命令提示符中执行)
2 mkdir C:\RAG_Evaluation
3 cd C:\RAG_Evaluation
4
5 # macOS / Linux 用户 (在终端中执行)
6 mkdir -p ~/RAG_Evaluation
7 cd ~/RAG_Evaluation
```

2.4 创建 Python 虚拟环境

虚拟环境是 Python 开发中的最佳实践。它为当前项目创建一个独立的 Python 运行环境，安装的所有依赖库都只在这个环境中生效，不会影响系统中的其他 Python 项目。

注意

虽然虚拟环境不是必须的，但强烈建议使用。它可以避免不同项目之间的依赖冲突，也便于项目的迁移和部署。

```
1 # Windows 系统
2 python -m venv venv
3 venv\Scripts\activate
4
5 # macOS / Linux 系统
6 python3 -m venv venv
```

```
7 source venv/bin/activate  
8  
9 # 激活成功后，命令行提示符前会出现 (venv) 标识
```

2.5 安装依赖库

评测脚本需要用到多个 Python 库，包括网络请求、数据处理、可视化等。为了方便管理，我们将所有依赖写入 `requirements.txt` 文件。

在项目目录下创建 `requirements.txt` 文件，内容如下：

```
1 requests>=2.28.0  
2 pandas>=1.5.0  
3 numpy>=1.23.0  
4 matplotlib>=3.6.0  
5 seaborn>=0.12.0  
6 tqdm>=4.64.0  
7 openpyxl>=3.0.0  
8 python-dotenv>=1.0.0  
9 jieba>=0.42.1
```

各依赖库的用途说明：

- `requests`: 用于调用 Dify API 进行检索
- `pandas`: 用于处理评测集和结果数据
- `numpy`: 用于数值计算
- `matplotlib` 和 `seaborn`: 用于生成可视化图表
- `tqdm`: 用于显示进度条
- `openpyxl`: 用于读写 Excel 文件
- `python-dotenv`: 用于加载环境变量配置
- `jieba`: 用于中文分词（构建评测集时使用）

执行以下命令安装所有依赖：

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

提示

如果下载速度较慢，可以使用国内镜像源加速：

```
1 pip install -r requirements.txt -i https://pypi.tuna.tsinghua.edu.cn/  
    simple
```

2.6 获取 Dify API 配置

运行评测脚本需要配置 Dify 的 API 访问凭证。你需要从 Dify 后台获取两项信息：API Key 和知识库 ID。

2.6.1 获得 API Key

API Key 是访问 Dify API 的身份凭证。获取步骤如下：

1. 登录 Dify 控制台（本地部署版本或云端版本均可）
2. 点击左下角的「设置」按钮
3. 在设置页面中找到「API 密钥」选项
4. 点击「创建新密钥」按钮
5. 系统会生成一个新的 API Key，将其复制保存。注意：API Key 只会显示一次，请妥善保管

2.6.2 获得知识库 ID

每个知识库都有一个唯一的 ID。如果你想对比不同分块策略的效果，需要分别获取每个知识库的 ID。

获取步骤如下：

1. 进入 Dify 控制台的「知识库」页面
2. 点击进入你要评测的知识库
3. 查看浏览器地址栏中的 URL
4. URL 格式类似于：`https://xxx.dify.ai/datasets/abc123def456/documents`
5. 其中红色部分 abc123def456 就是该知识库的 ID

2.6.3 配置环境变量文件

为了安全地管理敏感配置信息，我们使用 `.env` 文件存储 API Key 和知识库 ID。在项目目录下创建 `.env` 文件，内容如下：

```
1 # Dify API 基础配置
2 DIFY_API_BASE=https://api.dify.ai/v1
3 DIFY_API_KEY=在此处粘贴你的API Key
4
5 # 知识库 ID 配置
```

```
6 # 可以配置多个知识库，用于对比不同分块策略的效果  
7  
8 # 使用"通用分块"策略的知识库  
9 DATASET_ID_GENERAL=在此处粘贴通用分块知识库的 ID  
10  
11 # 使用"父子分块"策略的知识库  
12 DATASET_ID_PARENT_CHILD=在此处粘贴父子分块知识库的 ID  
13  
14 # 使用"QA分块"策略的知识库  
15 DATASET_ID_QA=在此处粘贴 QA 分块知识库的 ID
```

警告

安全提醒：.env 文件包含 API Key 等敏感信息，切勿将其提交到公开的代码仓库。如果你使用 Git 进行版本控制，请确保将 .env 添加到 .gitignore 文件中。

3 评测集构建

评测集是整个评测流程的基础和核心。本节将详细介绍评测集的设计原则、数据格式，以及如何高效地构建评测集。

3.1 评测集的重要性

评测集本质上是一组”问题-标准答案”的对应关系。每条数据包含一个用户可能会问的问题，以及这个问题的正确答案所在的文档位置。

可以把评测集理解为一份”考试试卷”：我们用这份试卷来测试知识库检索系统的能力。试卷的质量直接决定了评测结果的可信度。

一份高质量的评测集应该满足以下标准：

- 覆盖面广**：评测集中的问题应该覆盖知识库的主要内容领域。如果知识库涵盖请假、报销、入职、社保等多个主题，评测集也应该包含这些主题的问题。
- 问法多样**：针对同一个知识点，应该设计多种不同的问法。例如，”年假有几天”和”我可以休多少天年假”问的是同一件事，但表述方式不同。多样化的问法可以测试系统对语义理解的鲁棒性。
- 标注准确**：每个问题都必须准确标注其正确答案所在的文档 ID（必须）和具体段落（可选）。标注错误会导致评测结果失真。
- 规模适中**：评测集的规模需要在统计意义和人工成本之间取得平衡。对于初步评测，50-100 条数据就足够看出趋势；对于正式的效果评估，建议准备 300-500 条数据。

3.2 评测集数据格式

评测集以 Excel (.xlsx) 或 CSV 格式存储，每行代表一条测试数据。标准格式包含以下字段：

字段名	数据类型	说明
id	整数	唯一标识符，从 1 开始递增
query	字符串	用户问题，即测试时用于检索的输入
gold_doc_id	字符串	标准答案所在文档的 ID (从 Dify 后台获取)
gold_chunk_text	字符串	标准答案所在的段落内容 (可选，用于人工复核)
category	字符串	问题所属的业务分类，便于按类别统计效果
difficulty	字符串	问题难度等级：easy / medium / hard

表 2: 评测集标准字段说明

3.3 评测集示例

以人事制度知识库为例，下面是一份评测集的示例数据：

id	query	gold_doc_id	gold_chunk_text	category
1	年假有多少天	doc_001	员工入职满一年，可享受 5 天带薪年假...	请假
2	怎么申请报销	doc_002	报销流程：1. 填写报销单 2. 报销部门审批...	报销
3	试用期多长时间	doc_003	试用期一般为 3 个月，特殊岗位不超过 6 个月	入职
4	社保什么时候交	doc_004	公司在员工入职次月 15 日前办理社保	社保
5	请假需要提前多久	doc_001	请假需提前 3 个工作日提交申请	请假

表 3: 评测集示例（人事制度场景）

从这个示例可以看出评测集的核心逻辑：建立“问题”与“答案来源”之间的对应关系。评测时，系统会检查检索结果中是否包含 gold_doc_id 指定的文档。

3.4 评测集构建工具

手工逐条填写评测集效率较低，特别是当知识库文档较多时。下面提供一个辅助脚本，可以从知识库中自动提取候选问题，大幅提升构建效率。

该脚本的工作原理是：遍历知识库中的所有文档和分段，基于规则提取可能的问题（如标题、FAQ 条目等），生成候选评测集。你只需要对生成的候选数据进行人工审核和筛选即可。

创建 `build_evaluation_set.py` 文件，内容如下：

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 评测集构建工具
5 用于从知识库文档中生成候选问题，辅助人工标注
6 """
7
8 import os
9 import json
10 import pandas as pd
11 from dotenv import load_dotenv
12 import requests
13 from tqdm import tqdm
14
15 # 加载环境变量
16 load_dotenv()
17
18 class EvaluationSetBuilder:
19     """评测集构建器"""
20
21     def __init__(self):
22         self.api_base = os.getenv('DIFY_API_BASE', 'https://api.dify.ai/v1')
23
24         self.api_key = os.getenv('DIFY_API_KEY')
25         self.headers = {
26             'Authorization': f'Bearer {self.api_key}',
27             'Content-Type': 'application/json'
28         }
29
30     def get_dataset_documents(self, dataset_id: str) -> list:
31         """
32             获取知识库中的所有文档列表
33
34             Args:
35                 dataset_id: 知识库 ID
36
37         
```

```

36     Returns:
37         文档列表
38     """
39
40     url = f"{self.api_base}/datasets/{dataset_id}/documents"
41     all_documents = []
42     page = 1
43
44     while True:
45         params = {'page': page, 'limit': 20}
46         response = requests.get(url, headers=self.headers, params=
47 params)
48
49         if response.status_code != 200:
50             print(f"获取文档列表失败: {response.text}")
51             break
52
53         data = response.json()
54         documents = data.get('data', [])
55
56         if not documents:
57             break
58
59         all_documents.extend(documents)
60
61         if not data.get('has_more', False):
62             break
63
64         page += 1
65
66     print(f"共获取到 {len(all_documents)} 个文档")
67     return all_documents
68
69
70     def get_document_segments(self, dataset_id: str, document_id: str) ->
71 list:
72     """
73         获取文档的所有分段
74
75     Args:
76         dataset_id: 知识库 ID
77         document_id: 文档 ID
78
79     Returns:
80         分段列表
81     """
82
83     url = f"{self.api_base}/datasets/{dataset_id}/documents/{document_id}/segments"

```

```

document_id}/segments"
79     all_segments = []
80     page = 1
81
82     while True:
83         params = {'page': page, 'limit': 100}
84         response = requests.get(url, headers=self.headers, params=
85         params)
86
86         if response.status_code != 200:
87             print(f"获取分段失败: {response.text}")
88             break
89
90         data = response.json()
91         segments = data.get('data', [])
92
93         if not segments:
94             break
95
96         all_segments.extend(segments)
97
98         if not data.get('has_more', False):
99             break
100
101     page += 1
102
103     return all_segments
104
105 def extract_candidate_questions(self, segment_text: str) -> list:
106     """
107         从文本段落中提取候选问题
108         基于规则的简单提取，可以替换为更智能的方法
109
110     Args:
111         segment_text: 段落文本
112
113     Returns:
114         候选问题列表
115     """
116
117     import jieba
118
119     questions = []
120
121     # 规则1: 如果段落包含"问："或"Q："，直接提取
122     if '问：' in segment_text or 'Q:' in segment_text or 'Q: ' in

```

```

segment_text:
    lines = segment_text.split('\n')
    for line in lines:
        if line.startswith('问：') or line.startswith('Q：') or line.startswith('Q: '):
            q = line.replace('问：', '').replace('Q：', '').replace('Q: ', '').strip()
            if q:
                questions.append(q)

# 规则2：提取标题类内容作为候选问题
lines = segment_text.split('\n')
for line in lines:
    line = line.strip()
    # 匹配标题格式
    if line and len(line) < 50:
        # 去除序号
        import re
        clean_line = re.sub(r'^[\d一二三四五六七八九十]+[\.\、\s]+', '',
                           line)
        if clean_line and len(clean_line) > 4:
            # 转换为问句
            if not clean_line.endswith('?') and not clean_line.endswith('？'):
                questions.append(f"{clean_line}是什么")
                questions.append(f"如何{clean_line}")

return questions[:5] # 每个段落最多返回5个候选问题

def build_candidate_set(self, dataset_id: str, output_file: str = 'candidate_questions.xlsx'):
    """
    构建候选评测集

    Args:
        dataset_id: 知识库ID
        output_file: 输出文件名
    """
    print("开始构建候选评测集...")

    # 获取所有文档
    documents = self.get_dataset_documents(dataset_id)

    candidates = []

```

```

161     for doc in tqdm(documents, desc="处理文档"):
162         doc_id = doc['id']
163         doc_name = doc.get('name', 'unknown')
164
165         # 获取文档分段
166         segments = self.get_document_segments(dataset_id, doc_id)
167
168         for seg in segments:
169             seg_id = seg.get('id', '')
170             seg_text = seg.get('content', '')
171
172             if not seg_text:
173                 continue
174
175             # 提取候选问题
176             questions = self.extract_candidate_questions(seg_text)
177
178             for q in questions:
179                 candidates.append({
180                     'id': len(candidates) + 1,
181                     'query': q,
182                     'gold_doc_id': doc_id,
183                     'gold_doc_name': doc_name,
184                     'gold_segment_id': seg_id,
185                     'gold_chunk_text': seg_text[:200], # 截取前200字符
186                     'category': '', # 待人工填写
187                     'difficulty': '', # 待人工填写
188                     'is_valid': '' , # 待人工确认 (Y/N)
189                 })
190
191         # 保存到Excel
192         df = pd.DataFrame(candidates)
193         df.to_excel(output_file, index=False, engine='openpyxl')
194
195         print(f"候选评测集已保存到: {output_file}")
196         print(f"共生成 {len(candidates)} 条候选问题")
197         print("请人工审核并标注 is_valid、category、difficulty 列")
198
199         return df
200
201
202 def create_empty_template(output_file: str = 'evaluation_set_template.xlsx'):
203     """
204     创建空的评测集模板

```

```

205
206     Args:
207         output_file: 输出文件名
208
209     """
210
211     template = pd.DataFrame({
212         'id': [1, 2, 3],
213         'query': ['示例问题1: 年假有多少天', '示例问题2: 怎么申请报销', '示例问题3: 试用期多长'],
214         'gold_doc_id': ['doc_001', 'doc_002', 'doc_003'],
215         'gold_chunk_text': ['员工入职满一年可享受5天带薪年假', '报销流程: 填写报销单后提交审批', '试用期一般为3个月'],
216         'category': ['请假', '报销', '入职'],
217         'difficulty': ['easy', 'medium', 'easy'],
218     })
219
220     template.to_excel(output_file, index=False, engine='openpyxl')
221     print(f"评测集模板已创建: {output_file}")
222     print("请按模板格式填写您的评测数据")
223
224
225 if __name__ == '__main__':
226     import argparse
227
228     parser = argparse.ArgumentParser(description='评测集构建工具')
229     parser.add_argument('--action', choices=['template', 'build'], default='template',
230                         help='操作类型: template-创建空模板, build-从知识库构建')
231     parser.add_argument('--dataset-id', type=str, help='知识库ID (build模式需要) ')
232     parser.add_argument('--output', type=str, default='evaluation_set.xlsx',
233                         help='输出文件名')
234
235     args = parser.parse_args()
236
237     if args.action == 'template':
238         create_empty_template(args.output)
239     elif args.action == 'build':
240         if not args.dataset_id:
241             print("错误: build模式需要指定 --dataset-id")
242             exit(1)
243         builder = EvaluationSetBuilder()
244         builder.build_candidate_set(args.dataset_id, args.output)

```

3.5 脚本使用方法

构建工具提供两种使用方式，可以根据实际情况选择。

3.5.1 方式一：创建空白模板

如果你希望完全手工填写评测集，可以先生成一个包含标准格式的空白模板：

```
1 python build_evaluation_set.py --action template --output my_evaluation_set.xlsx
```

执行后会生成一个 Excel 文件，其中包含示例数据和正确的列格式。你可以删除示例数据，按照格式填入自己的评测数据。

3.5.2 方式二：从知识库自动提取候选问题

如果知识库中的文档较多，推荐使用自动提取功能：

```
1 python build_evaluation_set.py --action build --dataset-id 你的知识库 ID --output candidates.xlsx
```

该命令会自动连接 Dify API，遍历知识库中的所有文档和分段，基于规则提取候选问题。提取的候选数据会保存到指定的 Excel 文件中。

3.6 人工审核与标注

自动提取的候选问题只是初步筛选的结果，不能直接用于评测。你需要进行人工审核，确保每条数据的质量。

审核流程如下：

1. 打开生成的候选评测集 Excel 文件
2. 逐条检查 `query` 列中的问题。判断标准是：这个问题是否像真实用户会问的？表述是否自然？如果不合适，在 `is_valid` 列填写 N
3. 对于合适的问题，补充填写 `category`（业务分类）和 `difficulty`（难度等级）字段
4. 核对 `gold_doc_id` 是否正确。这是最重要的一步，标注错误会直接影响评测结果的准确性
5. 审核完成后，筛选出 `is_valid = Y` 的记录，另存为最终的评测集文件

提示

标注规范（标注口径）非常重要。在开始标注前，需要明确以下规则，确保标注的一致性：

- **问题粒度**：一个问题应该对应一个明确的知识点，避免过于笼统或包含多个子问题
- **命中判定**：只要 TopK 检索结果中包含 `gold_doc_id` 指定的文档，即判定为命中
- **同义表达**：允许使用同义词或不同表述方式，但核心概念必须一致
- **多答案情况**：如果一个问题有多个正确答案来源，可以用英文逗号分隔多个文档 ID

4 执行检索评测

完成评测集构建后，就可以开始正式的检索评测了。本节提供完整的评测脚本代码，支持单次评测和批量对比评测。

4.1 核心评测模块

下面是核心评测脚本的完整代码。该脚本实现了以下功能：

- 加载评测集数据
- 调用 Dify API 执行检索
- 计算各项评测指标（Recall@K、Precision@K、F1@K、MRR）
- 保存详细结果和汇总指标

创建 `rag_evaluator.py` 文件，内容如下：

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 RAG 知识库检索评测工具
5 支持 Dify 知识库的召回率、精确率、F1、MRR 等指标评测
6 """
7
8 import os
9 import json
10 import time
```

```

11 import requests
12 import pandas as pd
13 import numpy as np
14 from typing import List, Dict, Tuple, Optional
15 from dataclasses import dataclass, field
16 from dotenv import load_dotenv
17 from tqdm import tqdm
18 from datetime import datetime
19
20 # 加载环境变量
21 load_dotenv()
22
23
24 @dataclass
25 class RetrievalResult:
26     """单次检索结果"""
27     query: str
28     retrieved_docs: List[Dict] # 检索到的文档列表
29     gold_doc_ids: List[str] # 标准答案文档ID列表
30     is_hit: bool = False # 是否命中
31     hit_rank: int = -1 # 首次命中的排名 (-1表示未命中)
32     latency_ms: float = 0 # 检索耗时 (毫秒)
33
34
35 @dataclass
36 class EvaluationMetrics:
37     """评测指标"""
38     total_queries: int = 0
39     hit_count: int = 0
40     recall_at_k: float = 0.0
41     precision_at_k: float = 0.0
42     f1_at_k: float = 0.0
43     mrr: float = 0.0
44     avg_latency_ms: float = 0.0
45     hit_distribution: Dict[int, int] = field(default_factory=dict) # 各排名命中数分布
46
47
48 class DifyRetriever:
49     """Dify 知识库检索器"""
50
51     def __init__(self, api_base: str = None, api_key: str = None):
52         """
53             初始化检索器

```

```

55     Args:
56         api_base: API 基础 URL
57         api_key: API 密钥
58     """
59
60     self.api_base = api_base or os.getenv('DIFY_API_BASE', 'https://api.dify.ai/v1')
61
62     self.api_key = api_key or os.getenv('DIFY_API_KEY')
63
64     if not self.api_key:
65         raise ValueError("未配置 DIFY_API_KEY, 请在 .env 文件中设置")
66
67     self.headers = {
68         'Authorization': f'Bearer {self.api_key}',
69         'Content-Type': 'application/json'
70     }
71
72
73     def retrieve(self, dataset_id: str, query: str, top_k: int = 5,
74                 score_threshold: float = 0.0) -> Tuple[List[Dict], float]:
75         """
76         执行检索
77
78         Args:
79             dataset_id: 知识库 ID
80             query: 查询问题
81             top_k: 返回结果数量
82             score_threshold: 分数阈值
83
84         Returns:
85             (检索结果列表, 耗时毫秒)
86         """
87
88         url = f"{self.api_base}/datasets/{dataset_id}/retrieve"
89
90         payload = {
91             "query": query,
92             "retrieval_model": {
93                 "search_method": "semantic_search", # 语义检索
94                 "reranking_enable": False, # 是否启用重排
95                 "top_k": top_k,
96                 "score_threshold_enabled": score_threshold > 0,
97                 "score_threshold": score_threshold
98             }
99         }
100
101         start_time = time.time()

```

```

99     try:
100         response = requests.post(url, headers=self.headers, json=
101 payload, timeout=30)
102         latency_ms = (time.time() - start_time) * 1000
103
104         if response.status_code != 200:
105             print(f"检索失败: {response.status_code} - {response.text}")
106
107         return [], latency_ms
108
109
110     data = response.json()
111     records = data.get('records', [])
112
113     # 标准化结果格式
114     results = []
115     for i, record in enumerate(records):
116         results.append({
117             'rank': i + 1,
118             'document_id': record.get('segment', {}).get(
119             'document_id', ''),
119             'segment_id': record.get('segment', {}).get('id', ''),
120             'content': record.get('segment', {}).get('content', '')
121             ,
122             'score': record.get('score', 0),
123             'document_name': record.get('segment', {}).get(
124             'document', {}).get('name', ''),
125             })
126
127     return results, latency_ms
128
129
130     except Exception as e:
131         latency_ms = (time.time() - start_time) * 1000
132         print(f"检索异常: {str(e)}")
133         return [], latency_ms
134
135     def retrieve_with_rerank(self, dataset_id: str, query: str, top_k: int
136 = 5,
137                             rerank_model: str = "bge-reranker-base") ->
138     Tuple[List[Dict], float]:
139         """
140             执行带重排的检索
141
142             Args:
143                 dataset_id: 知识库 ID
144                 query: 查询问题

```

```

137     top_k: 返回结果数量
138     rerank_model: 重排模型名称
139
140     Returns:
141         (检索结果列表, 耗时毫秒)
142     """
143     url = f"{self.api_base}/datasets/{dataset_id}/retrieve"
144
145     payload = {
146         "query": query,
147         "retrieval_model": {
148             "search_method": "semantic_search",
149             "reranking_enable": True,
150             "reranking_model": {
151                 "reranking_provider_name": "local",
152                 "reranking_model_name": rerank_model
153             },
154             "top_k": top_k,
155             "score_threshold_enabled": False
156         }
157     }
158
159     start_time = time.time()
160
161     try:
162         response = requests.post(url, headers=self.headers, json=payload, timeout=60)
163         latency_ms = (time.time() - start_time) * 1000
164
165         if response.status_code != 200:
166             print(f"检索失败: {response.status_code} - {response.text}")
167     )
168
169     return [], latency_ms
170
171     data = response.json()
172     records = data.get('records', [])
173
174     results = []
175     for i, record in enumerate(records):
176         results.append({
177             'rank': i + 1,
178             'document_id': record.get('segment', {}).get('document_id', ''),
179             'segment_id': record.get('segment', {}).get('id', ''),
180             'content': record.get('segment', {}).get('content', '')
181         })
182
183     return results, latency_ms

```

```

,
        'score': record.get('score', 0),
        'document_name': record.get('segment', {}).get('
document', {}).get('name', ''),
    })

182
    return results, latency_ms

183
184
except Exception as e:
    latency_ms = (time.time() - start_time) * 1000
    print(f"检索异常: {str(e)}")
    return [], latency_ms

189
190
191 class RAGEvaluator:
192     """RAG 检索评测器"""
193
194     def __init__(self, retriever: DifyRetriever):
195         """
196             初始化评测器
197
198             Args:
199                 retriever: 检索器实例
200
201             self.retriever = retriever
202             self.results: List[RetrievalResult] = []
203
204     def load_evaluation_set(self, file_path: str) -> pd.DataFrame:
205         """
206             加载评测集
207
208             Args:
209                 file_path: 评测集文件路径 (支持xlsx、csv)
210
211             Returns:
212                 评测集DataFrame
213
214             if file_path.endswith('.xlsx'):
215                 df = pd.read_excel(file_path, engine='openpyxl')
216             elif file_path.endswith('.csv'):
217                 df = pd.read_csv(file_path)
218             else:
219                 raise ValueError(f"不支持的文件格式: {file_path}")
220
221             # 验证必要列

```

```

222     required_columns = ['query', 'gold_doc_id']
223     for col in required_columns:
224         if col not in df.columns:
225             raise ValueError(f"评测集缺少必要列: {col}")
226
227     print(f"加载评测集: {len(df)} 条记录")
228     return df
229
230 def evaluate(self, dataset_id: str, evaluation_set: pd.DataFrame,
231             top_k: int = 5, use_rerank: bool = False,
232             rerank_model: str = None) -> EvaluationMetrics:
233     """
234     执行评测
235
236     Args:
237         dataset_id: 知识库 ID
238         evaluation_set: 评测集 DataFrame
239         top_k: TopK 值
240         use_rerank: 是否使用重排
241         rerank_model: 重排模型 (use_rerank=True 时需要)
242
243     Returns:
244         评测指标
245     """
246     self.results = []
247     hit_count = 0
248     total_rr = 0 # 用于计算MRR
249     total_latency = 0
250     hit_distribution = {i: 0 for i in range(1, top_k + 1)}
251
252     print(f"\n开始评测...")
253     print(f"知识库 ID: {dataset_id}")
254     print(f"TopK: {top_k}")
255     print(f"使用重排: {use_rerank}")
256     print(f"评测集大小: {len(evaluation_set)}")
257     print("-" * 50)
258
259     for idx, row in tqdm(evaluation_set.iterrows(), total=len(
260         evaluation_set), desc="评测进度"):
261         query = str(row['query'])
262
263         # 处理 gold_doc_id (可能是单个或多个, 用逗号分隔)
264         gold_doc_ids = [doc_id.strip() for doc_id in str(row['
265         gold_doc_id']).split(',')]
```

```

265     # 执行检索
266     if use_rerank and rerank_model:
267         retrieved_docs, latency = self.retriever.
268         retrieve_with_rerank(
269             dataset_id, query, top_k, rerank_model
270         )
271     else:
272         retrieved_docs, latency = self.retriever.retrieve(
273             dataset_id, query, top_k)
274
275     total_latency += latency
276
277     # 判断是否命中
278     is_hit = False
279     hit_rank = -1
280
281     for doc in retrieved_docs:
282         doc_id = doc.get('document_id', '')
283         if doc_id in gold_doc_ids:
284             is_hit = True
285             hit_rank = doc['rank']
286             break
287
288     if is_hit:
289         hit_count += 1
290         total_rr += 1.0 / hit_rank
291         hit_distribution[hit_rank] = hit_distribution.get(hit_rank,
292             0) + 1
293
294     # 记录结果
295     result = RetrievalResult(
296         query=query,
297         retrieved_docs=retrieved_docs,
298         gold_doc_ids=gold_doc_ids,
299         is_hit=is_hit,
300         hit_rank=hit_rank,
301         latency_ms=latency
302     )
303     self.results.append(result)
304
305     # 避免请求过快
306     time.sleep(0.1)
307
308     # 计算指标
309     total_queries = len(evaluation_set)

```

```

307     recall = hit_count / total_queries if total_queries > 0 else 0
308     precision = hit_count / (top_k * total_queries) if total_queries >
309     0 else 0
310     f1 = 2 * precision * recall / (precision + recall) if (precision +
311     recall) > 0 else 0
312     mrr = total_rr / total_queries if total_queries > 0 else 0
313     avg_latency = total_latency / total_queries if total_queries > 0
314     else 0
315
316     metrics = EvaluationMetrics(
317         total_queries=total_queries,
318         hit_count=hit_count,
319         recall_at_k=recall,
320         precision_at_k=precision,
321         f1_at_k=f1,
322         mrr=mrr,
323         avg_latency_ms=avg_latency,
324         hit_distribution=hit_distribution
325     )
326
327     return metrics
328
329
330     def get_detailed_results(self) -> pd.DataFrame:
331         """
332             获取详细结果
333
334             Returns:
335                 详细结果DataFrame
336
337             """
338
339         data = []
340
341         for r in self.results:
342             data.append({
343                 'query': r.query,
344                 'gold_doc_ids': ','.join(r.gold_doc_ids),
345                 'is_hit': r.is_hit,
346                 'hit_rank': r.hit_rank if r.hit_rank > 0 else 'N/A',
347                 'latency_ms': round(r.latency_ms, 2),
348                 'top1_doc_id': r.retrieved_docs[0]['document_id'] if r.
349                 retrieved_docs else '',
350                 'top1_score': round(r.retrieved_docs[0]['score'], 4) if r.
351                 retrieved_docs else '',
352                 'top1_content': r.retrieved_docs[0]['content'][:100] if r.
353                 retrieved_docs else '',
354             })
355
356         return pd.DataFrame(data)

```

```

346
347     def save_results(self, metrics: EvaluationMetrics, output_dir: str,
348                      config_name: str = "default"):
349         """
350             保存评测结果
351
352         Args:
353             metrics: 评测指标
354             output_dir: 输出目录
355             config_name: 配置名称 (用于区分不同实验)
356         """
357
358         os.makedirs(output_dir, exist_ok=True)
359         timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
360
361         # 保存指标摘要
362         summary = {
363             'config_name': config_name,
364             'timestamp': timestamp,
365             'total_queries': metrics.total_queries,
366             'hit_count': metrics.hit_count,
367             'recall_at_k': round(metrics.recall_at_k, 4),
368             'precision_at_k': round(metrics.precision_at_k, 4),
369             'f1_at_k': round(metrics.f1_at_k, 4),
370             'mrr': round(metrics.mrr, 4),
371             'avg_latency_ms': round(metrics.avg_latency_ms, 2),
372             'hit_distribution': metrics.hit_distribution
373         }
374
375         summary_file = os.path.join(output_dir, f"metrics_{config_name}_{timestamp}.json")
376         with open(summary_file, 'w', encoding='utf-8') as f:
377             json.dump(summary, f, ensure_ascii=False, indent=2)
378             print(f"指标摘要已保存: {summary_file}")
379
380         # 保存详细结果
381         detailed_df = self.get_detailed_results()
382         detailed_file = os.path.join(output_dir, f"detailed_{config_name}_{timestamp}.xlsx")
383         detailed_df.to_excel(detailed_file, index=False, engine='openpyxl')
384         print(f"详细结果已保存: {detailed_file}")
385
386
387
388     def print_metrics(metrics: EvaluationMetrics, config_name: str = ""):

```

```

389 """
390     打印评测指标
391
392     Args:
393         metrics: 评测指标
394         config_name: 配置名称
395     """
396     print("\n" + "=" * 50)
397     if config_name:
398         print(f"配置: {config_name}")
399         print("=" * 50)
400     print(f"总查询数:      {metrics.total_queries}")
401     print(f"命中数:        {metrics.hit_count}")
402     print(f"Recall@K:      {metrics.recall_at_k:.4f} ({metrics.recall_at_k
403           *100:.2f}%)")
404     print(f"Precision@K:    {metrics.precision_at_k:.4f} ({metrics.
405           precision_at_k*100:.2f}%)")
406     print(f"F1@K:          {metrics.f1_at_k:.4f}")
407     print(f"MRR:            {metrics.mrr:.4f}")
408     print(f"平均延迟:       {metrics.avg_latency_ms:.2f} ms")
409     print("-" * 50)
410     print("命中排名分布:")
411     for rank, count in sorted(metrics.hit_distribution.items()):
412         if count > 0:
413             print(f"  Rank {rank}: {count} ({count/metrics.total_queries
414           *100:.1f}%)")
415     print("=" * 50)
416
417
418 if __name__ == '__main__':
419     import argparse
420
421     parser = argparse.ArgumentParser(description='RAG 知识库检索评测工具')
422     parser.add_argument('--dataset-id', type=str, required=True, help='知识
423     库 ID')
424     parser.add_argument('--eval-set', type=str, required=True, help='评测集
425     文件路径')
426     parser.add_argument('--top-k', type=int, default=5, help='TopK值')
427     parser.add_argument('--use-rerank', action='store_true', help='是否使用
428     重排')
429     parser.add_argument('--rerank-model', type=str, default='bge-reranker-
430     base', help='重排模型')
431     parser.add_argument('--output-dir', type=str, default='./results', help
432     ='输出目录')
433     parser.add_argument('--config-name', type=str, default='default', help=

```

```

'配置名称')

426
427     args = parser.parse_args()
428
429     # 初始化
430     retriever = DifyRetriever()
431     evaluator = RAGEvaluator(retriever)
432
433     # 加载评测集
434     eval_set = evaluator.load_evaluation_set(args.eval_set)
435
436     # 执行评测
437     metrics = evaluator.evaluate(
438         dataset_id=args.dataset_id,
439         evaluation_set=eval_set,
440         top_k=args.top_k,
441         use_rerank=args.use_rerank,
442         rerank_model=args.rerank_model if args.use_rerank else None
443     )
444
445     # 打印结果
446     print_metrics(metrics, args.config_name)
447
448     # 保存结果
449     evaluator.save_results(metrics, args.output_dir, args.config_name)

```

4.2 运行评测

脚本支持多种参数配置，可以灵活地进行不同场景的评测。

4.2.1 基础评测命令

最简单的评测方式是指定知识库 ID、评测集路径和 TopK 值：

```

1 python rag_evaluator.py \
2   --dataset-id 你的知识库ID \
3   --eval-set evaluation_set.xlsx \
4   --top-k 5 \
5   --config-name "general_chunk" \
6   --output-dir ./results

```

参数说明：

- **--dataset-id**: 要评测的知识库 ID
- **--eval-set**: 评测集文件路径

- `--top-k`: 检索返回的结果数量，常用值为 3、5、10
- `--config-name`: 本次评测的配置名称，用于区分不同实验
- `--output-dir`: 结果输出目录

评测完成后，结果会保存在 `./results` 目录中，包括指标汇总（JSON 格式）和详细结果（Excel 格式）。

4.2.2 启用重排器的评测

如果你想评估重排器（Reranker）对检索效果的提升，可以添加 `--use-rerank` 参数：

```

1 python rag_evaluator.py \
2   --dataset-id 你的知识库 ID \
3   --eval-set evaluation_set.xlsx \
4   --top-k 5 \
5   --use-rerank \
6   --rerank-model bge-reranker-base \
7   --config-name "general_with_rerank" \
8   --output-dir ./results

```

通过对比启用和不启用重排器的评测结果，可以量化重排器带来的效果提升。

5 批量对比评测

在实际应用中，我们通常需要对比多种配置的效果。例如，对比三种分块策略（通用、父子、QA），同时测试多个 TopK 值（3、5、10），还要看重排器的影响。如果逐个运行，效率很低。

本节提供的批量评测脚本可以自动遍历所有配置组合，一次性完成全部评测，并生成汇总对比结果。

5.1 批量评测脚本

创建 `batch_evaluation.py` 文件，内容如下：

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 批量对比评测脚本
5 支持多知识库、多TopK、是否重排等多维度对比
6 """
7

```

```

8 import os
9 import json
10 import pandas as pd
11 from datetime import datetime
12 from dotenv import load_dotenv
13 from rag_evaluator import DifyRetriever, RAGEvaluator, print_metrics,
   EvaluationMetrics
14
15 load_dotenv()
16
17
18 def run_batch_evaluation(config: dict) -> dict:
19     """
20         运行批量评测
21
22     Args:
23         config: 评测配置, 包含:
24             - evaluation_set_path: 评测集路径
25             - datasets: 知识库配置列表
26             - top_k_list: TopK值列表
27             - use_rerank_options: 是否使用重排选项列表
28             - output_dir: 输出目录
29
30     Returns:
31         所有评测结果
32     """
33
34     retriever = DifyRetriever()
35     evaluator = RAGEvaluator(retriever)
36
37     # 加载评测集
38     eval_set = evaluator.load_evaluation_set(config['evaluation_set_path'])
39
40     all_results = []
41
42     # 遍历所有配置组合
43     for dataset_config in config['datasets']:
44         dataset_id = dataset_config['id']
45         dataset_name = dataset_config['name']
46
47         for top_k in config['top_k_list']:
48             for use_rerank in config['use_rerank_options']:
49                 config_name = f"{dataset_name}_top{top_k}"
50                 if use_rerank:
51                     config_name += "_rerank"

```

```

52     print(f"\n{'='*60}")
53     print(f"正在评测: {config_name}")
54     print(f"{'='*60}")

55     # 执行评测
56     metrics = evaluator.evaluate(
57         dataset_id=dataset_id,
58         evaluation_set=eval_set,
59         top_k=top_k,
60         use_rerank=use_rerank,
61         rerank_model=config.get('rerank_model', 'bge-reranker-
62 base'))
63     )

64     # 打印结果
65     print_metrics(metrics, config_name)

66     # 保存单次结果
67     evaluator.save_results(metrics, config['output_dir'],
68 config_name)

69     # 收集结果
70     all_results.append({
71         'config_name': config_name,
72         'dataset_name': dataset_name,
73         'dataset_id': dataset_id,
74         'top_k': top_k,
75         'use_rerank': use_rerank,
76         'total_queries': metrics.total_queries,
77         'hit_count': metrics.hit_count,
78         'recall': metrics.recall_at_k,
79         'precision': metrics.precision_at_k,
80         'f1': metrics.f1_at_k,
81         'mrr': metrics.mrr,
82         'avg_latency_ms': metrics.avg_latency_ms
83     })

84     # 保存汇总结果
85     summary_df = pd.DataFrame(all_results)
86     timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
87     summary_file = os.path.join(config['output_dir'], f"summary_{timestamp}
88 .xlsx")
89     summary_df.to_excel(summary_file, index=False, engine='openpyxl')
90     print(f"\n汇总结果已保存: {summary_file}")
91
92
93

```

```

94     # 保存 JSON 格式
95     json_file = os.path.join(config['output_dir'], f"summary_{timestamp}.json")
96     with open(json_file, 'w', encoding='utf-8') as f:
97         json.dump(all_results, f, ensure_ascii=False, indent=2)
98     print(f"JSON 结果已保存: {json_file}")
99
100    return all_results
101
102
103 if __name__ == '__main__':
104     # 评测配置
105     config = {
106         # 评测集路径
107         'evaluation_set_path': 'evaluation_set.xlsx',
108
109         # 要对比的知识库（不同分块策略）
110         'datasets': [
111             {
112                 'id': os.getenv('DATASET_ID_GENERAL', 'your_general_dataset_id'),
113                 'name': 'general' # 通用分块
114             },
115             {
116                 'id': os.getenv('DATASET_ID_PARENT_CHILD', 'your_parent_child_dataset_id'),
117                 'name': 'parent_child' # 父子分块
118             },
119             {
120                 'id': os.getenv('DATASET_ID_QA', 'your_qa_dataset_id'),
121                 'name': 'qa' # QA 分块
122             },
123         ],
124
125         # TopK 值列表
126         'top_k_list': [3, 5, 10],
127
128         # 是否使用重排选项
129         'use_rerank_options': [False, True],
130
131         # 重排模型
132         'rerank_model': 'bge-reranker-base',
133
134         # 输出目录
135         'output_dir': './results'

```

```

136     }
137
138     # 运行批量评测
139     results = run_batch_evaluation(config)
140
141     # 打印最终汇总
142     print("\n" + "="*80)
143     print("评测完成！最终汇总：")
144     print("="*80)
145
146     df = pd.DataFrame(results)
147     print(df.to_string(index=False))

```

6 可视化与报告生成

数据表格虽然包含了完整的评测结果，但不够直观。通过可视化图表，可以更清晰地展示不同配置之间的效果差异，便于做出决策。

本节提供的可视化脚本可以生成以下图表：

- **召回率对比柱状图**: 直观展示不同分块策略在各 TopK 值下的召回率
- **指标热力图**: 以矩阵形式展示所有配置组合的评测结果
- **延迟对比图**: 展示不同配置的检索响应时间
- **综合雷达图**: 在一张图中对比多个维度的指标
- **Markdown 报告**: 自动生成包含结论建议的文字报告

6.1 可视化脚本代码

创建 `visualization.py` 文件，内容如下：

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 评测结果可视化
5 生成对比图表和分析报告
6 """
7
8 import os
9 import json
10 import pandas as pd
11 import numpy as np

```

```

12 import matplotlib.pyplot as plt
13 import seaborn as sns
14 from datetime import datetime
15
16 # 设置中文字体
17 plt.rcParams['font.sans-serif'] = ['SimHei', 'Arial Unicode MS', 'DejaVu
    Sans']
18 plt.rcParams['axes.unicode_minus'] = False
19
20 # 设置绘图风格
21 sns.set_style("whitegrid")
22 plt.rcParams['figure.figsize'] = (12, 8)
23 plt.rcParams['figure.dpi'] = 150
24
25
26 def load_summary_data(json_file: str) -> pd.DataFrame:
27     """
28         加载汇总数据
29
30     Args:
31         json_file: JSON格式的汇总文件路径
32
33     Returns:
34         DataFrame
35     """
36     with open(json_file, 'r', encoding='utf-8') as f:
37         data = json.load(f)
38     return pd.DataFrame(data)
39
40
41 def plot_recall_comparison(df: pd.DataFrame, output_dir: str):
42     """
43         绘制召回率对比图
44
45     Args:
46         df: 评测结果DataFrame
47         output_dir: 输出目录
48     """
49     fig, axes = plt.subplots(1, 2, figsize=(14, 6))
50
51     # 图1: 不同分块策略的召回率对比 (按TopK分组)
52     ax1 = axes[0]
53
54     # 准备数据: 不使用重排的结果
55     df_no_rerank = df[df['use_rerank'] == False]

```

```

56
57     pivot_data = df_no_rerank.pivot(index='top_k', columns='dataset_name',
58                                     values='recall')
59
60     x = np.arange(len(pivot_data.index))
61     width = 0.25
62
63     colors = ['#2ecc71', '#3498db', '#e74c3c']
64
65     for i, col in enumerate(pivot_data.columns):
66         bars = ax1.bar(x + i*width, pivot_data[col], width, label=col,
67                         color=colors[i % len(colors)])
68         # 添加数值标签
69         for bar, val in zip(bars, pivot_data[col]):
70             ax1.text(bar.get_x() + bar.get_width()/2, bar.get_height() +
71                      0.01,
72                     f'{val:.2%}', ha='center', va='bottom', fontsize=9)
73
74     ax1.set_xlabel('Top K', fontsize=12)
75     ax1.set_ylabel('Recall@K', fontsize=12)
76     ax1.set_title('不同分块策略的召回率对比 (无重排)', fontsize=14,
77                   fontweight='bold')
78     ax1.set_xticks(x + width)
79     ax1.set_xticklabels([f'Top{k}' for k in pivot_data.index])
80     ax1.legend(title='分块策略')
81     ax1.set_ylim(0, 1.1)
82
83     # 图2: 重排前后对比
84     ax2 = axes[1]
85
86     # 选择 Top5 的数据进行重排前后对比
87     df_top5 = df[df['top_k'] == 5]
88
89     datasets = df_top5['dataset_name'].unique()
90     x = np.arange(len(datasets))
91     width = 0.35
92
93     recall_no_rerank = df_top5[df_top5['use_rerank'] == False].set_index(
94         'dataset_name')['recall']
95     recall_with_rerank = df_top5[df_top5['use_rerank'] == True].set_index(
96         'dataset_name')['recall']
97
98     bars1 = ax2.bar(x - width/2, [recall_no_rerank.get(d, 0) for d in
99                         datasets],
100                            width, label='无重排', color='#3498db')

```

```

94 bars2 = ax2.bar(x + width/2, [recall_with_rerank.get(d, 0) for d in
95 datasets],
96 width, label='有重排', color='#e74c3c')
97
98 # 添加数值标签
99 for bars in [bars1, bars2]:
100     for bar in bars:
101         ax2.text(bar.get_x() + bar.get_width()/2, bar.get_height() +
102 0.01,
103             f'{bar.get_height():.2%}', ha='center', va='bottom',
104             fontsize=9)
105
106 ax2.set_xlabel('分块策略', fontsize=12)
107 ax2.set_ylabel('Recall@5', fontsize=12)
108 ax2.set_title('重排前后召回率对比 (Top5)', fontsize=14, fontweight='bold')
109 ax2.set_xticks(x)
110 ax2.set_xticklabels(datasets)
111 ax2.legend()
112 ax2.set_ylim(0, 1.1)
113
114 plt.tight_layout()
115
116 output_file = os.path.join(output_dir, 'recall_comparison.png')
117 plt.savefig(output_file, bbox_inches='tight')
118 plt.close()
119 print(f"召回率对比图已保存: {output_file}")
120
121
122 def plot_metrics_heatmap(df: pd.DataFrame, output_dir: str):
123     """
124     绘制指标热力图
125
126     Args:
127         df: 评测结果DataFrame
128         output_dir: 输出目录
129     """
130     fig, axes = plt.subplots(1, 2, figsize=(14, 6))
131
132     # 热力图1: Recall
133     ax1 = axes[0]
134     df_no_rerank = df[df['use_rerank'] == False]
135     pivot_recall = df_no_rerank.pivot(index='dataset_name', columns='top_k',
136             values='recall')

```

```

134     sns.heatmap(pivot_recall, annot=True, fmt=' .2%', cmap='YlGnBu', ax=ax1,
135                 vmin=0, vmax=1, cbar_kws={'label': 'Recall'})
136     ax1.set_title('召回率热力图（无重排）', fontsize=14, fontweight='bold')
137     ax1.set_xlabel('Top K')
138     ax1.set_ylabel('分块策略')
139
140     # 热力图2: MRR
141     ax2 = axes[1]
142     pivot_mrr = df_no_rerank.pivot(index='dataset_name', columns='top_k',
143                                    values='mrr')
144
145     sns.heatmap(pivot_mrr, annot=True, fmt=' .3f', cmap='YlOrRd', ax=ax2,
146                 vmin=0, vmax=1, cbar_kws={'label': 'MRR'})
147     ax2.set_title('MRR热力图（无重排）', fontsize=14, fontweight='bold')
148     ax2.set_xlabel('Top K')
149     ax2.set_ylabel('分块策略')
150
151     plt.tight_layout()
152
153     output_file = os.path.join(output_dir, 'metrics_heatmap.png')
154     plt.savefig(output_file, bbox_inches='tight')
155     plt.close()
156     print(f"指标热力图已保存: {output_file}")
157
158 def plot_latency_comparison(df: pd.DataFrame, output_dir: str):
159     """
160         绘制延迟对比图
161
162     Args:
163         df: 评测结果DataFrame
164         output_dir: 输出目录
165     """
166     fig, ax = plt.subplots(figsize=(10, 6))
167
168     # 按配置名排序
169     df_sorted = df.sort_values(['dataset_name', 'top_k', 'use_rerank'])
170
171     # 创建配置标签
172     labels = []
173     for _, row in df_sorted.iterrows():
174         rerank_str = '+rerank' if row['use_rerank'] else ''
175         labels.append(f'{row["dataset_name"]}\nTop{row["top_k"]}{rerank_str}')
176

```

```

177 colors = ['#3498db' if not r else '#e74c3c' for r in df_sorted['use_rerank']]
178
179 bars = ax.barih(range(len(labels)), df_sorted['avg_latency_ms'], color=colors)
180
181 ax.set_yticks(range(len(labels)))
182 ax.set_yticklabels(labels, fontsize=9)
183 ax.set_xlabel('平均延迟 (ms)', fontsize=12)
184 ax.set_title('检索延迟对比', fontsize=14, fontweight='bold')
185
186 # 添加数值标签
187 for bar, val in zip(bars, df_sorted['avg_latency_ms']):
188     ax.text(bar.get_width() + 5, bar.get_y() + bar.get_height()/2,
189             f'{val:.0f}ms', ha='left', va='center', fontsize=9)
190
191 # 添加图例
192 from matplotlib.patches import Patch
193 legend_elements = [Patch(facecolor='#3498db', label='无重排'),
194                     Patch(facecolor='#e74c3c', label='有重排')]
195 ax.legend(handles=legend_elements, loc='lower right')
196
197 plt.tight_layout()
198
199 output_file = os.path.join(output_dir, 'latency_comparison.png')
200 plt.savefig(output_file, bbox_inches='tight')
201 plt.close()
202 print(f"延迟对比图已保存: {output_file}")
203
204
205 def plot_comprehensive_comparison(df: pd.DataFrame, output_dir: str):
206     """
207         绘制综合对比图（类似用户提供的示例图）
208
209     Args:
210         df: 评测结果DataFrame
211         output_dir: 输出目录
212     """
213     fig, axes = plt.subplots(2, 2, figsize=(14, 12))
214
215     # 选择不使用重排的数据
216     df_no_rerank = df[df['use_rerank'] == False]
217
218     # 图1: 召回率折线图
219     ax1 = axes[0, 0]

```

```

220     for dataset in df_no_rerank['dataset_name'].unique():
221         data = df_no_rerank[df_no_rerank['dataset_name'] == dataset]
222         ax1.plot(data['top_k'], data['recall'], marker='o', linewidth=2,
223                   markersize=8, label=dataset)
224         ax1.set_xlabel('Top K', fontsize=12)
225         ax1.set_ylabel('Recall@K', fontsize=12)
226         ax1.set_title('召回率随TopK变化趋势', fontsize=14, fontweight='bold')
227         ax1.legend()
228         ax1.set_ylim(0, 1.05)
229         ax1.grid(True, alpha=0.3)
230
231 # 图2: F1分数对比
232 ax2 = axes[0, 1]
233 pivot_f1 = df_no_rerank.pivot(index='top_k', columns='dataset_name',
234                                values='f1')
235 pivot_f1.plot(kind='bar', ax=ax2, color=['#2ecc71', '#3498db', '#e74c3c'])
236 ax2.set_xlabel('Top K', fontsize=12)
237 ax2.set_ylabel('F1 Score', fontsize=12)
238 ax2.set_title('F1分数对比', fontsize=14, fontweight='bold')
239 ax2.legend(title='分块策略')
240 ax2.set_xticklabels([f'Top{k}' for k in pivot_f1.index], rotation=0)
241
242 # 图3: MRR对比
243 ax3 = axes[1, 0]
244 pivot_mrr = df_no_rerank.pivot(index='top_k', columns='dataset_name',
245                                 values='mrr')
246 pivot_mrr.plot(kind='bar', ax=ax3, color=['#9b59b6', '#f39c12', '#1abc9c'])
247 ax3.set_xlabel('Top K', fontsize=12)
248 ax3.set_ylabel('MRR', fontsize=12)
249 ax3.set_title('MRR (Mean Reciprocal Rank) 对比', fontsize=14,
250                 fontweight='bold')
251 ax3.legend(title='分块策略')
252 ax3.set_xticklabels([f'Top{k}' for k in pivot_mrr.index], rotation=0)
253
254 # 图4: 综合雷达图 (选择Top5数据)
255 ax4 = axes[1, 1]
256
257 df_top5 = df_no_rerank[df_no_rerank['top_k'] == 5]
258
259 categories = ['Recall', 'Precision', 'F1', 'MRR']
260
261 # 计算角度
262 angles = np.linspace(0, 2*np.pi, len(categories), endpoint=False).

```

```

        tolist()
260    angles += angles[:1]  # 闭合
261
262    ax4 = fig.add_subplot(2, 2, 4, polar=True)
263
264    colors = ['#2ecc71', '#3498db', '#e74c3c']
265    for i, (_, row) in enumerate(df_top5.iterrows()):
266        values = [row['recall'], row['precision'], row['f1'], row['mrr']]
267        values += values[:1]
268        ax4.plot(angles, values, 'o-', linewidth=2, color=colors[i % len(
269            colors)],
270                  label=row['dataset_name'])
271        ax4.fill(angles, values, alpha=0.1, color=colors[i % len(colors)])
272
273    ax4.set_xticks(angles[:-1])
274    ax4.set_xticklabels(categories)
275    ax4.set_title('Top5综合指标雷达图', fontsize=14, fontweight='bold', pad
276 =20)
277    ax4.legend(loc='upper right', bbox_to_anchor=(1.3, 1.0))
278
279    plt.tight_layout()
280
281    output_file = os.path.join(output_dir, 'comprehensive_comparison.png')
282    plt.savefig(output_file, bbox_inches='tight')
283    plt.close()
284    print(f"综合对比图已保存: {output_file}")
285
286
287    def generate_report(df: pd.DataFrame, output_dir: str):
288        """
289        生成文字报告
290
291        Args:
292            df: 评测结果DataFrame
293            output_dir: 输出目录
294        """
295
296        df_no_rerank = df[df['use_rerank'] == False]
297        df_with_rerank = df[df['use_rerank'] == True]
298
299        # 找出最佳配置
300        best_recall_idx = df_no_rerank['recall'].idxmax()
301        best_recall_config = df_no_rerank.loc[best_recall_idx]

```

```

302
303     report = f"""
304 # RAG 知识库检索评测报告
305
306 生成时间: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}
307
308 ## 评测概要
309
310 - 总评测配置数: {len(df)}
311 - 评测查询数: {df['total_queries'].iloc[0]}
312 - 分块策略: {', '.join(df['dataset_name'].unique())}
313 - TopK范围: {', '.join([f'Top{k}' for k in sorted(df['top_k'].unique())])}
314
315 ## 最佳配置
316
317 ### 召回率最高配置
318 - 配置: {best_recall_config['dataset_name']} + Top{best_recall_config['
    top_k']}
319 - Recall@K: {best_recall_config['recall']:.2%}
320 - MRR: {best_recall_config['mrr']:.4f}
321
322 ### MRR最高配置
323 - 配置: {best_mrr_config['dataset_name']} + Top{best_mrr_config['top_k']}
324 - Recall@K: {best_mrr_config['recall']:.2%}
325 - MRR: {best_mrr_config['mrr']:.4f}
326
327 ## 分块策略对比 (Top5, 无重排)
328
329 | 分块策略 | Recall@5 | Precision@5 | F1@5 | MRR | 平均延迟 |
330 | -----|-----|-----|-----|-----|-----|
331 """
332
333     df_top5_no_rerank = df_no_rerank[df_no_rerank['top_k'] == 5]
334     for _, row in df_top5_no_rerank.iterrows():
335         report += f"| {row['dataset_name']} | {row['recall']:.2%} | {row['
precision']:.4f} | {row['f1']:.4f} | {row['mrr']:.4f} | {row['
avg_latency_ms']:.0f}ms |\n"
336
337     # 重排效果分析
338     if len(df_with_rerank) > 0:
339         report += """
340 ## 重排效果分析 (Top5)
341
342 | 分块策略 | 无重排Recall | 有重排Recall | 提升 |
343 | -----|-----|-----|-----|

```

```

344 """
345     for dataset in df['dataset_name'].unique():
346         no_rerank = df_no_rerank[(df_no_rerank['dataset_name'] ==
347 dataset) & (df_no_rerank['top_k'] == 5)]
348         with_rerank = df_with_rerank[(df_with_rerank['dataset_name'] ==
349 dataset) & (df_with_rerank['top_k'] == 5)]
350
351         if len(no_rerank) > 0 and len(with_rerank) > 0:
352             recall_no = no_rerank['recall'].iloc[0]
353             recall_with = with_rerank['recall'].iloc[0]
354             improvement = recall_with - recall_no
355             report += f" | {dataset} | {recall_no:.2%} | {recall_with
356 :.2%} | {improvement:+.2%} |\n"
357
358 report += """
359 ## 结论与建议
360 """
361
362 # 自动生成建议
363 datasets = df_no_rerank['dataset_name'].unique()
364 recall_by_dataset = df_no_rerank.groupby('dataset_name')['recall'].mean()
365 best_dataset = recall_by_dataset.idxmax()
366
367 report += f"1. **推荐分块策略**: {best_dataset} (平均召回率最高: {recall_by_dataset[best_dataset]:.2%}) \n\n"
368
369 # TopK建议
370 recall_by_topk = df_no_rerank.groupby('top_k')['recall'].mean()
371 best_topk = recall_by_topk.idxmax()
372 report += f"2. **推荐TopK值**: Top{best_topk} (该TopK下平均召回率: {recall_by_topk[best_topk]:.2%}) \n\n"
373
374 # 重排建议
375 if len(df_with_rerank) > 0:
376     avg_improvement = (df_with_rerank['recall'].mean() - df_no_rerank[
377 df_no_rerank['top_k'].isin(df_with_rerank['top_k'].unique())]['recall'].mean())
378
379     if avg_improvement > 0.05:
380         report += f"3. **重排效果显著**: 平均提升 {avg_improvement:.2%}
381 , 建议启用重排器\n\n"
382
383     else:
384         report += f"3. **重排效果有限**: 平均提升仅 {avg_improvement
385 :.2%} , 可根据延迟要求决定是否启用\n\n"

```

```

379     report += """
380     """
381     """
382     报告由 RAG 评测工具自动生成
383     """
384
385     # 保存报告
386     report_file = os.path.join(output_dir, f"evaluation_report_{datetime.
387         now().strftime('%Y%m%d_%H%M%S')}.md")
388     with open(report_file, 'w', encoding='utf-8') as f:
389         f.write(report)
390     print(f"评测报告已保存: {report_file}")
391
392
393
394 def main(summary_json_path: str, output_dir: str = './results'):
395     """
396     主函数: 生成所有可视化图表和报告
397
398     Args:
399         summary_json_path: 汇总 JSON 文件路径
400         output_dir: 输出目录
401     """
402     os.makedirs(output_dir, exist_ok=True)
403
404     # 加载数据
405     df = load_summary_data(summary_json_path)
406
407     print("开始生成可视化图表...")
408
409     # 生成各类图表
410     plot_recall_comparison(df, output_dir)
411     plot_metrics_heatmap(df, output_dir)
412     plot_latency_comparison(df, output_dir)
413     plot_comprehensive_comparison(df, output_dir)
414
415     # 生成文字报告
416     report = generate_report(df, output_dir)
417
418     print("\n所有图表和报告已生成完成!")
419     print(f"输出目录: {output_dir}")
420
421
422 if __name__ == '__main__':

```

```
423 import argparse
424
425 parser = argparse.ArgumentParser(description='评测结果可视化工具')
426 parser.add_argument('--input', type=str, required=True, help='汇总 JSON
427 文件路径')
428 parser.add_argument('--output-dir', type=str, default='./results', help
429 ='输出目录')
430
431 args = parser.parse_args()
432
433 main(args.input, args.output_dir)
```

6.2 运行可视化脚本

可视化脚本需要两个输入：批量评测生成的 JSON 汇总文件，以及输出目录。

执行步骤：

```
1 # 步骤一：运行批量评测（如果尚未运行）
2 python batch_evaluation.py
3
4 # 步骤二：运行可视化脚本，生成图表和报告
5 # 注意将文件名替换为实际生成的 JSON 文件名
6 python visualization.py --input results/summary_20240101_120000.json --
    output-dir ./results
```

执行完成后，`results` 目录中将包含以下文件：

- `recall_comparison.png`: 召回率对比柱状图
- `metrics_heatmap.png`: 指标热力图
- `latency_comparison.png`: 延迟对比图
- `comprehensive_comparison.png`: 综合对比图（含雷达图）
- `evaluation_report_xxx.md`: Markdown 格式的评测报告

7 一键执行完整流程

前面介绍的评测流程涉及多个步骤和脚本。为了简化操作，本节提供一个整合脚本，可以一键完成从环境检查到图表生成的全部流程。

7.1 整合脚本

创建 run_evaluation.py 文件，内容如下：

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 一键运行完整评测流程
5 """
6
7 import os
8 import sys
9 from datetime import datetime
10 from dotenv import load_dotenv
11
12 load_dotenv()
13
14
15 def main():
16     print("=="*60)
17     print("RAG 知识库检索评测 - 完整流程")
18     print("=="*60)
19
20     # 检查必要文件
21     if not os.path.exists('.env'):
22         print("错误：未找到 .env 配置文件")
23         print("请创建 .env 文件并配置 DIFY_API_KEY 等参数")
24         sys.exit(1)
25
26     eval_set_file = 'evaluation_set.xlsx'
27     if not os.path.exists(eval_set_file):
28         print(f"未找到评测集文件: {eval_set_file}")
29         print("正在创建评测集模板...")
30         from build_evaluation_set import create_empty_template
31         create_empty_template(eval_set_file)
32         print(f"请编辑 {eval_set_file} 填入评测数据后重新运行")
33         sys.exit(0)
34
35     # 创建输出目录
36     timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
37     output_dir = f"./results_{timestamp}"
38     os.makedirs(output_dir, exist_ok=True)
39
40     print(f"\n输出目录: {output_dir}")
41
42     # 配置评测参数
```

```

43 config = {
44     'evaluation_set_path': eval_set_file,
45     'datasets': [
46         {'id': os.getenv('DATASET_ID_GENERAL'), 'name': 'general'},
47         {'id': os.getenv('DATASET_ID_PARENT_CHILD'), 'name': 'parent_child'},
48         {'id': os.getenv('DATASET_ID_QA'), 'name': 'qa'},
49     ],
50     'top_k_list': [3, 5, 10],
51     'use_rerank_options': [False, True],
52     'rerank_model': 'bge-reranker-base',
53     'output_dir': output_dir
54 }
55
56 # 过滤掉未配置的知识库
57 config['datasets'] = [d for d in config['datasets'] if d['id']]
58
59 if not config['datasets']:
60     print("错误：未配置任何知识库ID")
61     print("请在 .env 文件中配置 DATASET_ID_GENERAL 等参数")
62     sys.exit(1)
63
64 print(f"\n将评测以下知识库:")
65 for d in config['datasets']:
66     print(f" - {d['name']}: {d['id']}")
67
68 # 运行批量评测
69 print("\n" + "="*60)
70 print("开始批量评测...")
71 print("="*60)
72
73 from batch_evaluation import run_batch_evaluation
74 results = run_batch_evaluation(config)
75
76 # 找到生成的 JSON 文件
77 import glob
78 json_files = glob.glob(os.path.join(output_dir, 'summary_*.*json'))
79 if not json_files:
80     print("错误：未找到汇总 JSON 文件")
81     sys.exit(1)
82
83 json_file = sorted(json_files)[-1] # 取最新的
84
85 # 生成可视化图表
86 print("\n" + "="*60)

```

```

87 print("生成可视化图表...")
88 print("="*60)
89
90 from visualization import main as vis_main
91 vis_main(json_file, output_dir)
92
93 print("\n" + "="*60)
94 print("评测完成！")
95 print("="*60)
96 print(f"\n请查看输出目录: {output_dir}")
97 print("包含以下文件:")
98 for f in os.listdir(output_dir):
99     print(f" - {f}")
100
101
102 if __name__ == '__main__':
103     main()

```

7.2 快速上手指南

如果你希望尽快跑通整个流程，可以按照以下精简步骤操作：

1. 步骤一：创建项目目录并安装依赖

```

1 # 创建项目目录
2 mkdir RAG_Evaluation && cd RAG_Evaluation
3
4 # 安装所有依赖（单行命令）
5 pip install requests pandas numpy matplotlib seaborn tqdm openpyxl
    python-dotenv jieba
6

```

2. 步骤二：配置 API 访问凭证

在项目目录下创建 .env 文件，填入以下配置信息：

```

1 DIFY_API_BASE=https://api.dify.ai/v1
2 DIFY_API_KEY=替换为你的API Key
3 DATASET_ID_GENERAL=替换为通用分块知识库ID
4 DATASET_ID_PARENT_CHILD=替换为父子分块知识库ID
5 DATASET_ID_QA=替换为QA分块知识库ID
6

```

3. 步骤三：准备评测集

```
1 # 方式一：生成空白模板，手工填写
2 python build_evaluation_set.py --action template --output
   evaluation_set.xlsx
3
4 # 方式二：从知识库自动提取候选问题（需人工审核）
5 python build_evaluation_set.py --action build --dataset-id 你的知识库ID
   --output candidates.xlsx
6
```

4. 步骤四：执行评测

```
1 # 运行一键评测脚本
2 python run_evaluation.py
3
```

执行完成后，在 `results` 目录中查看评测结果、对比图表和分析报告。

8 常见问题与解决方案

本节汇总了使用过程中可能遇到的常见问题及其解决方法。

8.1 API 调用错误

错误信息	解决方案
401 Unauthorized	API Key 无效或已过期。请登录 Dify 后台，重新生成一个新的 API Key
403 Forbidden	权限不足。请检查该 API Key 是否有权限访问目标知识库
404 Not Found	资源不存在。请核实知识库 ID 是否正确拼写
429 Too Many Requests	请求频率过高。在脚本中增加 <code>time.sleep()</code> 的等待时间
请求超时	网络不稳定或知识库规模较大。尝试增大 <code>timeout</code> 参数值

表 4: API 错误代码及解决方案

8.2 评测结果异常

在分析评测结果时，可能会遇到以下异常情况：

- **召回率为 0**: 最可能的原因是评测集中的 `gold_doc_id` 填写错误。请登录 Dify 后台，核对实际的文档 ID 是否与评测集中的一致。
- **召回率达到 100%**: 虽然看起来是好事，但需要警惕评测集是否过于简单。建议检查评测集中的问题是否与知识库内容高度重复，适当增加一些难度较高的问题。
- **多次运行结果不一致**: 这是正常现象，可能由于 API 响应的随机性或评测集规模较小导致。建议多次运行取平均值，或增大评测集规模以获得更稳定的结果。

8.3 图表中文显示异常

如果生成的图表中，中文字符显示为方块或乱码，这是因为系统缺少中文字体。解决方法如下：

macOS 系统: 通常无需额外安装，系统自带中文字体。

Ubuntu/Linux 系统:

```
1 # 安装文泉驿中文字体
2 sudo apt install fonts-wqy-zenhei
```

Windows 系统: 下载并安装 SimHei（黑体）字体，双击字体文件即可安装。

安装字体后，需要修改 `visualization.py` 中的字体配置：

```
1 # Ubuntu 系统
2 plt.rcParams['font.sans-serif'] = ['WenQuanYi Zen Hei']
3
4 # Windows 系统
5 plt.rcParams['font.sans-serif'] = ['SimHei']
```

9 附录

9.1 项目目录结构

```
1 RAG_Evaluation/
2 |-- .env                      # 环境变量配置 (API Key 等)
3 |-- requirements.txt            # Python 依赖
4 |-- build_evaluation_set.py    # 评测集构建工具
5 |-- rag_evaluator.py           # 核心评测模块
6 |-- batch_evaluation.py        # 批量评测脚本
7 |-- visualization.py          # 可视化图表生成
```

```

8 |-- run_evaluation.py          # 一键运行脚本
9 |-- evaluation_set.xlsx       # 评测集文件
10 |-- results/                 # 评测结果输出目录
11   |-- metrics_xxx.json       # 指标 JSON
12   |-- detailed_xxx.xlsx      # 详细结果
13   |-- summary_xxx.xlsx       # 汇总结果
14   |-- recall_comparison.png  # 召回率对比图
15   |-- metrics_heatmap.png    # 热力图
16   |-- comprehensive_comparison.png # 综合对比图
17   |-- evaluation_report.md   # 评测报告

```

9.2 评测指标数学定义

本节给出各评测指标的严格数学定义，供需要深入了解的读者参考。

Recall@K（召回率）

召回率衡量的是：在所有测试问题中，有多少问题能在 Top K 检索结果中找到正确答案。

直观公式：

$$Recall@K = \frac{\text{Top K 中包含正确答案的问题数}}{\text{总问题数}}$$

严格数学定义：

$$Recall@K = \frac{|\{q : TopK(q) \cap Gold(q) \neq \emptyset\}|}{|Q|}$$

其中， Q 是测试问题集合， $TopK(q)$ 是问题 q 的 Top K 检索结果集合， $Gold(q)$ 是问题 q 的标准答案文档集合。

Precision@K（精确率）

精确率衡量的是：在所有检索返回的结果中，有多少是正确的。

$$Precision@K = \frac{\sum_{q \in Q} |TopK(q) \cap Gold(q)|}{K \times |Q|}$$

F1@K（F1 分数）

F1 分数是召回率和精确率的调和平均值。只有当两个指标都较高时，F1 分数才会高。

$$F1@K = \frac{2 \times Precision@K \times Recall@K}{Precision@K + Recall@K}$$

MRR（Mean Reciprocal Rank，平均倒数排名）

MRR 衡量的是正确答案在检索结果中出现的位置。正确答案排名越靠前，MRR 分数越高。

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{rank_q}$$

其中， $rank_q$ 是问题 q 的第一个正确答案在检索结果中的排名。

举例说明：

- 正确答案排在第 1 位，贡献 $\frac{1}{1} = 1.0$
- 正确答案排在第 2 位，贡献 $\frac{1}{2} = 0.5$
- 正确答案排在第 5 位，贡献 $\frac{1}{5} = 0.2$
- 未找到正确答案，贡献 0

9.3 参考资料

如需进一步学习 RAG 系统评测的相关知识，推荐以下资源：

- **Dify 官方文档**: <https://docs.dify.ai/>
包含知识库配置、API 使用等详细说明。
- **RAGAS 评测框架**: <https://github.com/explodinggradients/ragas>
开源的 RAG 评测工具，提供了更多评测指标和自动化评测能力。
- **LlamaIndex 评测指南**: <https://docs.llamaindex.ai/>
另一个流行的 RAG 框架的评测方法论和最佳实践。

本文档提供了 *Dify* 知识库召回率评测的完整流程和工具。

如有问题或建议，欢迎反馈。