第5章 RAG技术

随着生成式人工智能(Generative AI)的快速发展，检索增强生成(Retrieval-Augmented Generation，RAG)技术已成为解决大语言模型知识静态性、幻觉问题及领域适应性瓶颈的核心方案。本章系统解析RAG的技术框架、实现路径与行业实践：从基础理论(定义、发展脉络与核心价值)到系统架构(检索器、生成器与增强策略的协同机制)，从关键技术(文本向量化、混合检索优化、提示词工程)到实战部署(基于LangChain的工程实现、企业级性能调优与容错设计)。同时，结合金融、医疗等领域的真实案例，探讨RAG在知识更新效率、数据隐私保护与推理可解释性上的独特优势，并针对知识淹没、检索噪声等挑战提出技术破局思路，为开发者在效率与安全的平衡中提供方法论支撑。

【教学目标】

- 了解 RAG 技术的基本定义、发展历程及其对 LLM 能力的增强逻辑。
- 了解 RAG 在智能客服、知识问答、法律文书生成等场景中的典型应用模式。
- 了解 RAG 技术面临的核心挑战。
- 熟悉 RAG 系统的三阶段工作流程与核心组件功能解耦。
- 熟悉文本向量化与混合检索策略的技术原理。
- 掌握使用 LangChain 框架构建端到端 RAG 系统。

5.1 RAG 技术概述

本节以“技术原理→应用场景→落地价值”为主线，结合行业案例与实操演示，系统解析 RAG 技术的核心逻辑及其对生产生活的实际价值。内容设计贴合应用型学生特点，并注重理论联系实际进行讲解。

5.1.1 RAG 定义与实用价值

1. RAG 的定义

检索增强生成(Retrieval-Augmented Generation，RAG)是一种创新技术，它巧妙地将信息检索与文本生成能力融合。其核心理念如下：当大语言模型(LLM)需要回答用户问题时，它不再仅仅依赖自身记忆，而是会像一个研究者一样，首先通过检索模块从外部庞大的知识库中快速查找并获取相关信息片段。随后，这些检索到的准确资料会作为额外上下文，输入给生成模型。模型基于这些“最新查阅的资料”和原始问题，生成更加精确、可靠的答案。

通俗来说，RAG 赋予大语言模型一种“先查阅资料，再撰写回答”的能力，使其能像“学者”般严谨，而非仅凭记忆回答。

2. RAG 的实用价值

RAG 技术因其独特优势，有效解决了传统生成式 AI 面临的两大核心挑战。

1) 解决知识更新滞后问题

传统大型 AI 模型一旦训练完成，其知识便趋于固化，难以快速捕捉新发生的信息和行业动态(例如，某些模型可能仅掌握到 2023 年年末的数据)。这种“时间冻结”特性严重限制了它们在快速变化的现实世界中的应用。

RAG 通过解耦模型的“生成能力”与“知识存储”，构建了一个动态、可实时更新的知识体系。它将大模型的通用理解与外部知识库的最新信息相结合，实现“即插即用”。这意味着企业能迅速响应政策变化或技术迭代，其知识更新效率远超传统模型的微调方式。

2) 显著提升专业领域回答的准确性

大语言模型在生成内容时，常出现“幻觉”现象——即编造看似合理但实际错误的信息，这在专业领域尤为突出。例如，在复杂的医疗诊断中，未增强的大语言模型可能给出高错误率的建议，这源于其训练数据的广泛性与专业知识深度之间的矛盾。

RAG 通过引入三重校验机制，极大地提升了生成内容的可靠性：

(1) 权威知识锚定：强制模型从预设的专业文档库(如行业标准、官方指南)中提取信息，杜绝随意发挥，确保信息来源权威。

(2) 混合检索策略：结合语义向量检索(捕捉深层关联)与关键词检索(锁定精准术语)。例如，在医疗中可同时匹配疾病症状描述和其对应的国际疾病编码，提升检索的全面性和准确性。

(3) 提示词工程约束：通过结构化指令(如“严格依据××标准回答”)引导生成方向，确保输出内容严格遵循特定规则或要求，进一步提高精确性和合规性。

传统生成式 AI 与 RAG 增强系统对比如表 5.1 所示。

表 5.1 传统生成式 AI 与 RAG 增强系统对比

维　　度	传统生成式 AI	RAG 增强系统
知识更新周期	可能长达数月(依赖模型参数)	实时(分钟级索引更新)
专业领域准确性	错误率较高(依赖模型记忆)	外部知识驱动，错误率大幅降低
部署成本	微调在算力方面的消耗大	轻量级知识库维护
可解释性	黑箱操作，决策路径模糊	检索结果可溯源，符合审计要求

5.1.2 RAG 技术发展历程

检索增强生成(RAG)技术并非一夜建成，而是经历了从概念萌芽、模型确立，到持续优化，最终迈向多模态与智能体驱动的演进历程。

1. 早期探索与基础 RAG 阶段（2017—2020 年）

1) 萌芽阶段(2017—2019 年)

在 RAG 概念正式提出前，研究者已开始探索检索与生成结合。例如，Google Research 的 REALM 模型在预训练阶段就引入了检索机制，能高效地从大规模文本库(如百度百科)中检索相关文档，以此增强语言模型的理解能力。

2) RAG 模型正式提出(2020 年)

2020 年，Meta AI(当时的 Facebook AI Research)的团队正式发布了 RAG 模型，详细阐述其原理和优势。这标志着 RAG 技术的诞生，迅速引起广泛关注，并在开放域问答等任务上超越了当时的先进生成模型。

2. 优化、扩展与标准化阶段(2020—2022 年)

RAG 模型提出后，研究焦点转向性能提升和应用拓展。

1) 检索器优化

(1) DPR(Dense Passage Retriever，2020)：专为开放域问答设计，平衡了检索的精确性与效率。

(2) ANCE(Approximate Nearest Neighbor Negative Contrastive Learning，2021)：通过融合近似最近邻搜索和对比学习，大幅降低了大规模检索的延迟，是检索工程化的重要突破。

2) 生成器优化

(1) T5(Text-to-Text Transfer Transformer，2020)：统一了自然语言处理任务为“文本输入-文本输出”的范式，简化了多任务适配。

(2) 提示学习(Prompt Learning):通过精心设计的提示词,更有效地引导生成模型输出符合预期的答案。

(3) FiD(Fusion-in-Decoder):一种微调方法,让生成器能同时处理多个检索到的文档,并在解码时进行多文档间的注意力计算,综合考量信息。

3. 多模态 RAG 阶段(2022 年至今)

随着多模态大模型的兴起,RAG 技术也实现了跨越式发展,不再局限于文本,开始处理图像、视频、音频等多种数据类型。

(1) Flamingo(DeepMind,2022):作为视觉语言模型的代表,Flamingo 能同时接收图像和文本输入,并生成文本输出。它在多种图像和视频任务上表现出色,尤其在少样本学习方面展现了强大能力。

(2) OmniSearch:这是首个自适应多模态检索规划智能体。OmniSearch 模拟人类解决复杂问题的认知过程,能将复杂问题分解为一系列子问题,并根据实时检索反馈动态调整策略,在多模态理解与决策方面迈出重要一步。

5.1.3 RAG 应用场景

RAG 在不同层面所应对的挑战如下。

1) 行业痛点

(1) 知识的时效性问题(Knowledge Staleness)。

痛点:许多行业,如金融、法律、科技等,信息更新速度快,而预训练的大语言模型的知识截止日期通常较早,无法获取最新的信息,导致生成的内容可能过时或不准确。

RAG 解决方案:RAG 通过在生成答案之前实时检索最新的行业报告、新闻、研究论文等外部知识,确保模型能够利用最新的信息来回答用户的问题,提供更具时效性的答案。

(2) 领域知识的不足(Lack of Domain-Specific Knowledge)。

痛点:通用大语言模型虽然拥有广泛的知识,但在特定行业或企业的专业领域知识方面可能不足,难以回答需要深入专业知识的问题。

RAG 解决方案:RAG 可以接入行业特定的知识库,如产品文档、内部规程、专业术语表等。通过检索这些专业知识,模型能够生成更专业、更准确的回答,满足行业用户的特定需求。

(3) 生成内容的事实性问题(Hallucinations and Factual Errors)。

痛点:大语言模型有时会生成听起来合理但不真实的信息(即“幻觉”),这在需要高度准确性的行业(如医疗、法律)是不可接受的。

RAG 解决方案:RAG 通过检索外部的权威信息作为生成答案的依据,可以显著降低模型产生幻觉的风险。生成的答案有检索到的证据支持,提高了其可信度和准确性。

(4) 复杂问题的理解和回答(Understanding and Answering Complex Questions)。

痛点:某些行业的问题可能非常复杂,涉及多个知识点和推理步骤,通用大语言模型可能难以准确理解和生成全面的答案。

RAG 解决方案:RAG 可以通过对复杂问题进行分解,并检索多个相关的文档片段作为上下文,帮助大语言模型更好地理解问题的各个方面,并生成更完整、更深入的答案。

(5) 数据孤岛和信息分散(Data Silos and Information Fragmentation)。

痛点：许多企业内部的知识分散在不同的文档、数据库和系统中，员工难以快速找到所需的信息，影响工作效率。

RAG 解决方案：RAG 可以连接到各种不同的数据源，将分散的知识整合起来，并通过自然语言查询的方式，让员工能够更方便地获取所需的信息，打破信息孤岛。

(6) 定制化和个性化需求(Customization and Personalization Requirements)。

痛点：不同行业或企业有其特定的业务流程、术语和数据格式，通用大语言模型难以直接满足这些定制化的需求。

RAG 解决方案：RAG 可以根据特定行业或企业的需求，接入特定的知识库和数据，并进行相应的定制化开发，以更好地满足其个性化的应用场景。

2) 背景痛点

(1) 缺乏上下文的生成(Generation Without Sufficient Context)。

痛点：纯粹的生成模型在没有充分上下文的情况下，可能生成模糊、不相关或过于宽泛的答案。

RAG 解决方案：RAG 通过检索与用户查询相关的背景信息，为生成模型提供了更丰富的上下文，使其能够生成更具体、更相关的答案。

(2) 模型知识更新的成本和难度(Cost and Difficulty of Model Knowledge Updates)。

痛点：重新训练大语言模型以更新其知识库，需要巨大的计算资源和时间成本，且难以频繁进行。

RAG 解决方案：RAG 将知识更新的负担从模型本身转移到外部知识库。更新知识库中的文档比重新训练模型要高效和经济得多。

(3) 模型的可解释性和可追溯性(Interpretability and Traceability of Model Output)。

痛点：纯粹的生成模型的决策过程往往难以解释，用户难以判断生成答案的依据是否可靠。

RAG 解决方案：RAG 提供了生成答案的证据来源(即检索到的文档片段)，增强了模型输出的可解释性和可追溯性，用户可以验证答案的依据。

(4) 处理长文本的挑战(Challenges in Processing Long Texts)。

痛点：大语言模型的上下文窗口有限，难以直接处理和理解非常长的文档。

RAG 解决方案：RAG 可以将长文档分割成更小的语义单元(Chunk)，并检索与用户查询最相关的 Chunk 作为上下文，从而有效地处理长文本信息。

(5) 用户意图的准确理解(Accurate Understanding of User Intent)。

痛点：用户提出的问题可能存在歧义或隐含的意图，纯粹的检索或生成模型可能难以准确把握。

RAG 解决方案：RAG 结合了检索和生成的能力，可以通过检索相关信息来帮助模型更好地理解用户的真实意图，并生成更符合用户需求的答案。

由于 RAG 巧妙地将信息检索与文本生成模型相结合，不仅提升了生成内容的质量和可靠性，更在诸多行业展现出巨大的应用潜力。下面将深入探讨 RAG 的各种应用场景，尤其要关注最新的行业动态。

1. 智能客服与客户支持

在客户服务领域，RAG 技术的应用正在彻底改变传统的客服模式。传统的 AI 聊天机

器人往往依赖于预编程的回答，对于复杂或新颖的问题常常显得力不从心。而基于 RAG 的聊天机器人则不同，它们能够实时检索企业的知识库、产品文档、常见问题解答库及过往的客户交互记录等信息源，为客户提供准确且上下文相关的回复。

例如，在电子商务行业，客户在购物过程中可能会对产品的特性、使用方法、售后服务等方面提出各种问题。RAG 聊天机器人可以迅速从产品手册和售后政策中检索相关内容，并结合自然语言生成技术，以清晰易懂的方式回答客户的疑问。在金融领域，当客户咨询关于贷款产品、投资策略或账户操作等复杂问题时，RAG 技术能够确保客服系统提供合规且最新的信息，增强客户对金融机构的信任。

全球知名的商业信息和内容技术提供商汤森路透，就采用了基于 GPT-4 的 RAG 聊天机器人，帮助客户做出更明智的决策。实践证明，该聊天机器人高效且经济，有效减少了模型幻觉问题，为客户支持树立了新的标杆。

2. 医疗保健行业

在医疗保健领域，准确和及时的信息对于患者护理和医疗决策至关重要。RAG 技术在这一领域有着广泛的应用前景。

对于患者而言，他们可以通过 RAG 驱动的医疗咨询平台，询问关于疾病症状、治疗方案、药物副作用、康复时间等问题。平台能够检索权威的医学文献、临床指南、研究报告及患者教育资料，为患者提供详细且科学的解答，帮助患者更好地理解自身病情和治疗选择。对于医疗专业人员，RAG 技术可以成为强大的辅助工具。在诊断过程中，医生可以利用 RAG 系统检索类似病例、最新的医学研究成果及专家共识，为诊断和治疗方案的制定提供更多参考依据。在制定复杂疾病的治疗计划时，系统能够快速整合多源信息，帮助医生权衡不同治疗方案的利弊，做出更精准的决策。

3. 教育培训

在教育领域，RAG 技术为个性化学习和智能辅导带来了新的可能性。学生在学习过程中常常会遇到各种问题，无论是关于课程内容、作业难题还是考试复习。RAG 驱动的智能学习助手可以根据学生的问题，检索课程教材、学术论文、在线学习资源及过往的答疑记录，为学生提供详细的解答和指导。

例如，在高等教育中，学生在撰写论文时可能需要对特定主题进行深入研究。RAG 系统可以帮助学生快速检索相关的学术文献，并生成文献综述的初稿，引导学生进一步探索和分析。在职业培训中，学员可以通过 RAG 平台获取行业最新动态、技能操作指南及案例分析，加速学习进程，提升培训效果。

4. 知识管理与企业协作

在企业内部，知识管理是提高工作效率和促进创新的关键因素。RAG 技术可以帮助企业构建智能知识管理系统，员工能够通过自然语言查询，快速获取企业内部的各种文档、报告、项目经验、最佳实践等知识资产。

当员工需要了解公司的某项政策、查找特定项目的相关资料或者借鉴以往类似项目的经验时，RAG 系统能够迅速从企业知识库中检索出相关信息，并以易于理解的方式呈现给员工。这不仅节省了员工查找信息的时间，还促进了企业内部知识的共享和传承，提升了整体协作效率。

5. 内容创作与营销

对于内容创作者和营销人员来说，RAG 技术是提升工作效率和创作质量的有力工具。在内容创作过程中，创作者可以利用 RAG 系统检索行业最新趋势、市场数据、竞争对手动态及热门话题，为创作提供丰富的素材和灵感。

在制定营销文案时，RAG 系统可以分析过往成功营销案例，结合当前市场趋势和目标受众特点，生成具有吸引力的文案初稿。营销人员还可以通过 RAG 技术优化电子邮件营销活动、社交媒体内容策划及搜索引擎优化（SEO）策略，提高营销活动的针对性和效果。

6. 旅游与酒店业

在旅游和酒店行业，RAG 技术可以为游客提供个性化的旅行规划和信息服务。游客在计划旅行时，可以通过 RAG 聊天机器人咨询关于旅游目的地的景点介绍、美食推荐、住宿选择、交通指南等问题。

聊天机器人能够检索旅游攻略、酒店评价、当地活动信息及实时交通数据，为游客量身定制旅行计划，并提供实时更新的信息。在酒店预订过程中，客人可以通过 RAG 系统了解酒店的房型、设施、优惠活动及周边配套服务，帮助他们做出更合适的选择。

7. 制造业与工业互联网

在制造业和工业互联网领域，RAG 技术也开始展现出独特的价值。例如，在智能工厂中，当工人或技术人员在设备维护、生产流程优化等方面遇到问题时，可以借助 RAG 系统查询设备手册、维修记录、生产工艺标准等信息，快速解决问题，减少生产停机时间。

树根互联股份有限公司正在探索将 RAG 技术应用于智能客服、数字员工及工业机器人等领域，通过结合大模型技术和高效检索系统，推动工业互联网数字化转型服务的创新发展。

RAG 技术作为人工智能领域的一项重要创新，正在多个行业中发挥着重要作用，为各行业的数字化转型和智能化升级提供了强大的支持。随着技术的不断发展和完善，RAG 有望在更多领域实现更深入的应用，为人们的生活和工作带来更多便利和创新。

5.2 RAG 系统核心架构

5.2.1 RAG 系统三大核心组件

RAG 系统由 3 个核心组件构成：检索器（Retriever）、生成器（Generator）和知识库（Knowledge Base）。每个组件都扮演着重要的角色，共同协作以实现高质量的文本生成。

1. 检索器（Retriever）

检索器是 RAG 系统的第一个关键组件，负责从外部知识库中找到与用户查询最相关的信息。它的主要任务是高效、准确地识别并提取与给定问题或需求相关的知识片段。检索器的性能直接影响 RAG 系统最终生成的内容质量。

1）检索器的核心功能

检索器的核心功能如下：接收用户输入的自然语言查询，以理解其查询意图；通过预先构建的倒排索引、向量索引等常见索引结构加速知识查找过程；评估知识库中各文档或

知识片段与用户查询的相关程度；依据相关性得分对检索到的知识进行排序及筛选，挑出最相关部分；最后将筛选后的知识片段返回给生成器，作为生成文本的依据。

2）检索器的实现方式

检索器的实现方式多种多样，可以根据具体的应用场景和知识库的特点选择合适的方法。检索器的主要实现方式包括如下 3 种。

（1）基于关键词的检索：依赖于关键词匹配，通过查找用户查询中包含的关键词在知识库中出现的频率和位置来确定相关性。其优点在于实现简单、速度快，缺点是无法理解语义信息，且容易受到关键词歧义的影响，导致检索结果可能不准确。

（2）基于向量相似度的检索：将用户查询和知识库中的文档都转换为向量表示，然后通过计算向量之间的相似度来衡量相关性。其优点在于能够捕捉语义信息，对关键词的变体和同义词具有较好的适应性，检索结果更准确；缺点是需要预先训练或使用现成的词向量模型，计算量较大。

（3）混合检索：结合了基于关键词的检索和基于向量相似度的检索的优点，通过一定的策略将两种方法的检索结果进行融合，能够兼顾检索的速度和准确性，适用于复杂的应用场景。

3）检索器的评估指标

（1）准确率（Precision）：检索到的相关文档占所有检索到的文档的比例。

（2）召回率（Recall）：检索到的相关文档占所有知识库中相关文档的比例。

（3）F1 值：准确率和召回率的调和平均数，综合评价检索器的性能。

（4）平均精度均值（MAP）：衡量检索系统在多个查询下的平均检索精度。

2. 生成器（Generator）

生成器是 RAG 系统的第二个核心组件，负责根据检索器提供的知识片段和用户查询生成自然流畅的文本。生成器的主要任务是将检索到的信息整合到生成的文本中，确保内容的相关性、准确性和可读性。

1）生成器的核心功能

生成器的核心功能如下。

（1）接收用户查询和检索结果：接收用户输入的自然语言查询及检索器返回的相关知识片段。

（2）理解上下文信息：理解用户查询的意图及检索到的知识片段的含义。

（3）生成文本：根据用户查询和检索到的知识片段，生成自然流畅的文本，满足用户的需求。

（4）控制生成质量：控制生成文本的风格、长度、准确性等，确保生成的内容符合要求。

2）生成器的实现方式

生成器通常基于 Transformer 架构的大语言模型实现。这些模型经过预训练，具有强大的文本生成能力。例如，GPT-4 便是常用的生成器模型，其以强大的生成能力和通用性而闻名。

3）生成器的评估指标

BLEU 用于衡量生成文本与参考文本间的相似度，ROUGE 用于聚焦于衡量二者间的召回率，METEOR 则综合考量准确率与召回率，并对词序进行优化。最后还有人工评估方

式的指标，该指标主要通过人工评价生成文本在流畅性、相关性及准确性等方面的表现。

3. 知识库(Knowledge Base)

知识库是 RAG 系统的第三个核心组件，是存储和管理外部知识的场所。外部知识可以是任何形式的结构化或非结构化数据，例如，文档集合(如 PDF 文档、Word 文档等)、数据库(如关系数据库、非关系数据库等)、以图形结构表示知识的知识图谱等。

1) 知识库的核心功能

知识库的核心功能如下。

(1) 存储知识：存储各种形式的知识，并提供高效的访问接口。

(2) 知识更新：支持知识的更新和维护，确保知识库中的信息是最新的。

(3) 知识管理：提供知识管理工具，方便用户浏览、搜索、编辑和组织知识。

2) 知识库的类型

根据知识的结构化程度，知识库可以分为以下 3 类。

(1) 非结构化知识库：存储非结构化的文本数据，如文档集合、网页等。

(2) 半结构化知识库：存储半结构化的数据，如 XML、JSON 等。

(3) 结构化知识库：存储结构化的数据，如关系数据库、知识图谱等。

3) 知识库的构建方法

(1) 手动构建：通过人工录入或编辑知识，适用于规模较小、知识结构化的知识库。

(2) 自动构建：通过网络爬虫、文本挖掘等技术自动提取知识，适用于规模较大、知识来源广泛的知识库。

(3) 半自动构建：结合了手动构建和自动构建的优点，通过人工审核和修正自动提取的知识，提高知识的质量。

5.2.2 RAG 系统工作流程

RAG 系统的工作流程是一个高度标准化的“检索—生成”链路，涵盖从用户提问到答案输出的全生命周期。本节将通过流程图分解与实际案例解析，详细阐述其核心步骤与技术细节，其流程图如图 5.1 所示。

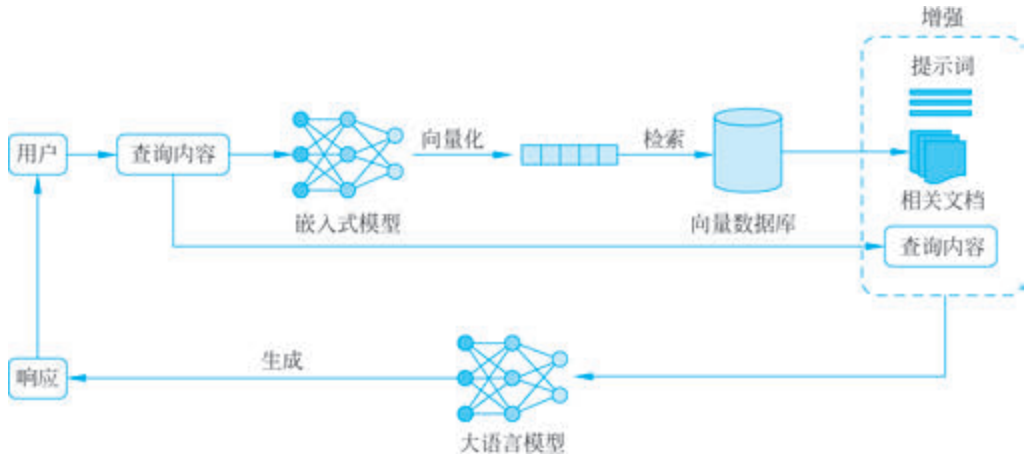


图 5.1 RAG 系统工作流程图

1. 全流程概述

1）输入与查询处理

（1）用户输入：接收用户的自然语言查询（如问题、指令或对话内容）。

（2）查询理解与重构：对输入进行语义解析，提取关键意图和实体（如命名实体识别、关键词提取），可能通过查询扩展或重写优化检索效果。

2）知识检索

文档/知识库准备：外部知识库可以是结构化数据（如数据库）、非结构化文本（如网页、PDF 文档）或实时更新的内容（如新闻、行业报告），数据尽可能经过预处理，用以支持高效检索。

3）上下文增强

（1）结果筛选与排序：根据相关性评分对检索结果去重、过滤低质量文档，保留 Top-K（如 3～5 条）最相关片段。

（2）上下文拼接：将检索到的文档片段与原始查询组合为增强输入（Prompt）。

4）生成

大语言模型推理：将增强后的内容输入生成模型（如 GPT-4），并基于检索内容生成最终回答。模型需具备对上下文的理解能力，并平衡检索内容与自身知识。

5）后处理与反馈（可选）

（1）结果验证：对生成内容进行事实性检查（Fact-Checking）或引用标注（如标记回答中引用的文档片段）。

（2）用户反馈闭环：收集用户对回答质量的评价，优化检索策略或调整生成参数（如动态调整 Top-K 值）。

2. 案例：医疗问答系统流程

1）用户提问

“妊娠期糖尿病患者应该如何控制血糖？”

2）预处理

实体抽取：妊娠期糖尿病、控制血糖。

问题向量化：通过 BGE-M3 模型生成 1024 维向量。

3）检索（Top-3）

《中国妊娠期糖尿病诊疗指南（2023）》：建议饮食控制与胰岛素治疗。

美国糖尿病协会（ADA）指南：推荐每日血糖监测。

某三甲医院临床研究：运动干预对血糖控制的影响。

4）生成

根据以下文档回答问题：

[文档 1]妊娠期糖尿病患者需每日监测空腹血糖……

[文档 2]ADA 建议碳水化合物摄入量控制在总热量的 40%以下……

问题：妊娠期糖尿病患者应该如何控制血糖？

5）生成结果

“建议通过饮食控制（如限制碳水化合物摄入）、每日血糖监测，并遵医嘱使用胰岛素治疗。[来源：指南 1，指南 2]”

6）后处理

NLI 验证：答案与文档无矛盾。

敏感信息过滤：无敏感内容。

5.3 RAG 关键技术解析

RAG 系统的核心技术围绕检索增强生成展开，其核心在于将大规模预训练语言模型与外部知识检索结合，解决传统生成模型的幻觉问题。本节将深入剖析 RAG 的关键技术模块。

5.3.1 文本向量化

文本向量化是 RAG 系统的基石，其目标是将自然语言转换为机器可理解的数值向量。根据向量表示方式的不同，主要分为稠密向量（Dense Vector）与稀疏向量（Sparse Vector）两类模型。两者的设计理念、技术特点与应用场景存在显著差异。

1. 稠密向量模型

稠密向量模型通过深度神经网络生成低维、连续、稠密的向量表示，捕捉文本的深层语义信息。

1）核心原理

神经网络架构：基于 Transformer 的编码器（如 BERT、RoBERTa）。

训练目标：通过掩码语言建模（MLM）或对比学习（Contrastive Learning），使语义相近的文本在向量空间中距离更近。

输出维度：通常为 128～1024 维，每个维度代表抽象的语义特征。

2）典型模型

在稠密向量模型应用中，不同模型各有优势，BERT 凭借双向编码捕捉上下文依赖，适用于通用语义匹配；Sentence-BERT 经句子优化，可快速计算相似度，助力检索与聚类；BGE-M3 支持多语言多粒度嵌入，服务跨语言检索；E5 基于对比学习，能增强语义区分，用于问答与重排序，如表 5.2 所示。

表 5.2 稠密向量模型

模型名称	特　点	适用场景
BERT	双向编码，捕捉上下文依赖	通用语义匹配
Sentence-BERT	针对句子的优化，支持快速相似度计算	检索、聚类
BGE-M3	多语言、多粒度（词/句/段）嵌入	跨语言检索
E5	基于对比学习，增强细粒度语义区分能力	问答、重排序（Rerank）

3）代码示例（Sentence-BERT）

```
import numpy as np
from sentence_transformers import SentenceTransformer
try:
    model = SentenceTransformer("sentence-transformers/all-mpnet-base-v2")
    sentences = ["糖尿病需控制饮食", "Insulin is used to treat diabetes."]
```

```
    embeddings = model.encode(sentences)
    print("向量维度:", embeddings.shape) # 输出:(2,768)
    cosine_similarity = np.dot(embeddings[0], embeddings[1]) / (np.linalg.norm(embeddings
[0]) * np.linalg.norm(embeddings[1]))
    print("余弦相似度:", cosine_similarity)
except Exception as e:
    print(f"出现错误: {e}")
```

余弦相似度公式的详细介绍如下。

对于两个向量 $\boldsymbol{A}$ 和 $\boldsymbol{B}$，余弦相似度定义为

$$\text{cosine\_similarity}(\boldsymbol{A},\boldsymbol{B}) = \frac{\boldsymbol{A}\cdot\boldsymbol{B}}{\|\boldsymbol{A}\|\ \|\boldsymbol{B}\|}$$

其中，$\boldsymbol{A}\cdot\boldsymbol{B}$ 是向量 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的点积(内积)，计算公式为

$$\boldsymbol{A}\cdot\boldsymbol{B} = \sum_{i=1}^{n}\boldsymbol{A}_i\boldsymbol{B}_i$$

其中，$\boldsymbol{A}_i$ 和 $\boldsymbol{B}_i$ 分别是向量 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的第 i 个分量，n 是向量的维度。

$\|\boldsymbol{A}\|$ 和 $\|\boldsymbol{B}\|$ 分别是向量 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的欧几里得范数(模)，计算公式为

$$\|\boldsymbol{A}\| = \sqrt{\sum_{i=1}^{n}\boldsymbol{A}_i^2}, \quad \|\boldsymbol{B}\| = \sqrt{\sum_{i=1}^{n}\boldsymbol{B}_i^2}$$

4) 优势与局限

稠密向量模型的优势是语义理解能力强，能够捕捉近义词、反义词以及上下文依赖关系，且泛化能力良好，适用于未登录词和多语言场景，但其存在计算资源需求高(模型参数量达百兆至千兆级)及可解释性差(向量维度难以直观解释)的局限。

2. 稀疏向量模型

稀疏向量模型基于词袋(Bag-of-Words)假设，通过统计方法生成高维、离散、稀疏的向量表示，强调显式关键词匹配。

1) 核心原理

稀疏向量模型的核心原理是通过词频统计方法(如 TF-IDF、BM25 等)构建高维稀疏向量，其维度与词典规模一致且大部分元素为零。

2) 典型模型

在稀疏向量模型的实践中，不同模型各有特性与适用场景。TF-IDF 简单快速，将词频统计用于关键词检索、文本分类；BM25 优化文档长度影响，适配搜索引擎与短文本检索；SPLADE 作为神经稀疏模型，可动态扩展查询词，助力自适应检索，如表 5.3 所示。

表 5.3 稀疏向量模型

模型名称	特点	适用场景
TF-IDF	简单快速，依赖词频统计	关键词检索、文本分类
BM25	优化文档长度影响，支持长尾词匹配	搜索引擎、短文本检索
SPLADE	神经稀疏模型，动态扩展查询词	领域自适应检索

3）代码示例（以 BM25 为例）

```
from rank_bm25 import BM25Okapi
import jieba
# 中文分词
corpus = ["糖尿病需控制饮食", "胰岛素用于治疗糖尿病"]
tokenized_corpus = [list(jieba.cut(doc)) for doc in corpus]
# 构建 BM25 模型
bm25 = BM25Okapi(tokenized_corpus)
query = list(jieba.cut("糖尿病治疗"))
scores = bm25.get_scores(query)
print("BM25 得分:", scores)
```

4）优势与局限

稀疏向量模型（如 BM25）的优势在于计算效率高，适合实时检索场景，且可解释性强，能通过关键词匹配得分直观反映相关性；其局限性在于语义捕捉能力较弱，难以处理近义词、多义词等语义变化（例如，“苹果”无法区分是指水果还是公司），同时高维稀疏向量会导致存储和计算资源消耗显著增加，影响大规模应用的效率。

3. 混合检索模型

为结合两类模型的优势，工业界常采用稠密＋稀疏的混合检索模型。

1）实现方式

并行检索：分别使用稠密模型和稀疏模型召回候选文档。

结果融合：通过加权得分、交叉编码器重排序或神经网络融合层，整合两种模型的召回结果。

2）代码示例（混合检索）

```
import numpy as np
# 假设已获得稠密检索得分 dense_scores 和 BM25 得分 bm25_scores
dense_scores = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
bm25_scores = np.array([10, 9, 8, 7, 6, 5, 4, 3, 2, 1])
hybrid_scores = 0.6 * dense_scores + 0.4 * bm25_scores          # 加权融合
top_indices = np.argsort(hybrid_scores)[::-1][:10]              # 取 Top-10
print(top_indices)
```

5.3.2 检索优化策略：混合检索（BM25＋向量）

混合检索通过结合稀疏检索（BM25）与稠密检索（向量模型）的优势，可以显著提升短文本场景下的召回率与准确率。

1. 混合检索的概念

混合检索模型通过并行执行稀疏检索（如 BM25）与稠密向量检索，分别获取关键词匹配和语义相关的候选结果，对两者分数归一化后按权重融合，最终去重并截取 Top-K 结果，在短文本场景下兼顾精准匹配与语义泛化能力，显著提升搜索效果。

2. 混合检索优势

（1）召回率提升：BM25 覆盖精确匹配结果，向量模型补充语义相关结果。

（2）准确率优化：通过分数融合过滤噪声，保留高置信度文档。

（3）领域鲁棒性：降低对单一模型的依赖，适配不同行业场景（医疗、法律等）。

5.3.3 生成器优化技巧：提示词工程与上下文融合

在 RAG 系统中，生成器的核心任务是将检索到的知识片段与用户问题深度融合，生成准确、流畅的答案。本节将深入探讨提示词工程与上下文融合两大关键技术，解析其优化策略与实现方法。

1）生成器的性能瓶颈

（1）信息过载：检索结果过长导致模型注意力分散。

（2）知识冲突：多文档间存在矛盾信息（如新旧指南差异）。

（3）语义偏差：生成答案偏离检索内容，产生“幻觉”。

2）优化目标

（1）精准性：确保答案与检索文档高度一致。

（2）可解释性：标注答案来源，增强可信度。

（3）可控性：通过参数约束生成风格（如专业/口语化）。

1. 提示词工程：引导模型聚焦关键信息

提示词工程通过设计输入模板，明确指导生成模型的行为，其核心在于结构化上下文组织与显式指令控制，提示词常用引导模板如表 5.4 所示。

表 5.4 常用引导模板

模板类型	结构示例	适用场景
指令式	“你是一名医生，根据以下医学指南回答问题：{context}。问题：{question}”	专业领域问答
问答式	“文档：[Doc1]A…[Doc2]B…问：{question}？答：”	多文档推理
思维链	“先分析原因，再给出建议。文档：{context}。问题：{question}。逐步思考：”	复杂逻辑推理
结构化输出	“生成 JSON：{'answer':'…','sources':[…]}”	API 调用

2. 上下文融合：知识的高效集成策略

上下文融合旨在将检索文档的关键信息无缝嵌入生成过程，解决长文本信息丢失问题，上下文融合主流技术方案如表 5.5 所示。

表 5.5 上下文融合主流技术方案

方法名称	原　理	优　缺　点
Concatenation	简单拼接问题和文档	易实现，但长文本效果差
FiD	各文档独立编码，解码时通过交叉注意力聚合信息	处理长文本优，计算成本高
FLARE	迭代生成：先预测答案需补充的知识，动态检索并融合	精准聚焦，延迟高
HierarchicalFusion	分层处理：先摘要文档，再融合摘要	平衡效率与效果，依赖摘要质量

3. 参数调优：平衡生成质量与多样性

在自然语言生成任务中，参数调优需在生成质量（准确性、连贯性）与多样性（创新性、丰富性）间寻求平衡。通过调整温度参数（如 temperature＝0.3 增强确定性，temperature＝1.0 提升随机性）、核采样阈值（top_p＝0.9 限制候选词范围）及重复惩罚（repetition_

penalty=1.2 抑制冗余),可对生成内容进行精细控制。例如,低温度结合束搜索(num_beams=4)可提升逻辑严谨性,适用于技术文档生成;而高温度配合随机采样(do_sample=True)则适合创意文本创作。需根据任务需求动态调整参数组合,如客服对话优先质量(低温度+低 top_p),故事续写侧重多样性(高温度+高 top_p),最终实现输出结果的最优权衡,如表 5.6 所示。

表 5.6 常见调优参数

参 数	作 用 域	典 型 值	影 响
temperature	采样策略	0.3(严谨)	值越低,输出越确定;值越高,越多样
top_p	核采样	0.9	从累计概率前 p 的 token 中采样,过滤离谱选项
max_length	生成长度	512	限制最大输出长度,防止无关内容
repetition_penalty	重复惩罚	1.2	抑制重复短语生成

示例代码如下(参数配置):

```
from transformers import GenerationConfig, AutoModelForCausalLM, AutoTokenizer
# 初始化 tokenizer 和 model
tokenizer = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelForCausalLM.from_pretrained("gpt2")
# 准备输入
input_text = "Once upon a time"
inputs = tokenizer(input_text, return_tensors="pt")
# 配置生成参数
generation_config = GenerationConfig(
    temperature=0.3,
    top_p=0.9,
    max_new_tokens=300,
    num_beams=4,
    do_sample=True,
    repetition_penalty=1.2,
    early_stopping=True
)
# 生成文本
output = model.generate( * * inputs, generation_config=generation_config)
# 解码输出
output_text = tokenizer.decode(output[0], skip_special_tokens=True)
print(output_text)
```

5.4 项目举例:医学领域 RAG 系统的实现

本节将以 LangChain 框架为核心,介绍实现一个完整的 RAG 系统,覆盖环境配置、代码解析,具体实验操作请参考配套资料“5.5 本章实验:基于 LangChain 的医学领域 RAG 系统的实现”Word 文件。本节使用系统的主要功能是构建一个医学领域的问答系统,以《中国糖尿病防治指南》为知识库,通过混合检索从知识库中获取相关信息,再利用医疗领域微调模型生成专业且严谨的答案,同时返回答案的来源文档,方便用户核实信息。

1. 项目模块概述

该项目借助 LangChain 的模块化设计,在一定程度上简化了 RAG 流程,核心组件及其

功能如下。

(1) 文档加载器：负责解析 PDF 格式的文档，代码中使用 PyPDFLoader 加载 PDF 文档。

(2) 文本分割器：按语义或长度对文档进行切分，代码中使用 RecursiveCharacterTextSplitter 对中文文本进行分割。

(3) 嵌入模型：生成文本的向量表示，代码采用 HuggingFaceEmbeddings。

(4) 向量数据库：存储和检索向量化后的文档，代码选用 FAISS。

(5) 检索器：执行混合检索与重排序，代码中综合运用了 BM25Retriever 和 FAISS 检索器，并通过 EnsembleRetriever 进行加权融合，还使用 CrossEncoderReranker 进行重排序。

(6) 生成模型：基于上下文生成答案，代码使用 HuggingFacePipeline 集成了医疗领域微调模型。

2. 参考代码

1) 知识库准备(document_preprocessing.py)

知识库的作用是把存储医学知识的 PDF 文档加载进来，并且按照一定的规则将其分割成合适大小的文本块。这样做的好处是，后续进行文本向量化和信息检索时会更加高效和精准。

```
# 加载 PDF 文档，借助 PyPDFLoader 类，它能将指定路径的 PDF 文件内容读取出来
    loader = PyPDFLoader(file_path)
    documents = loader.load()
# 中文文本分割(结合语义与长度)
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=512, chunk_overlap=64,
        separators=["\n\n", "\n", ".", "!", "?"],
        length_function=lambda x: len(list(jieba.cut(x))))
# 用于存储分割后的文本块，并遍历每个加载进来的文档
    chunks = []
    for doc in documents:
        # 对单个文档进行分块处理
        doc_chunks = text_splitter.split_documents([doc])
        # 将分块结果添加到总的文本块列表中
        chunks.extend(doc_chunks)
    return chunks
```

2) 数据库向量化(vector_database.py)

把准备好的文本数据转换为向量形式，并构建相应的向量数据库。向量数据库的作用在于能快速找到与查询内容最相似的文本。

```
# 初始化嵌入模型，使用指定的中文文本向量化模型
    embed_model = HuggingFaceEmbeddings(
        model_name="GanymedeNil/text2vec-large-chinese",
        model_kwargs={"device": "cpu"},
        encode_kwargs={"normalize_embeddings": True})
# 计算每个批次的大小，将文本合理地分批次处理且将文本划分为多个批次
    batch_size = max(1, len(texts) // 10)
    all_batches = [texts[i:i+batch_size] for i in range(0, len(texts), batch_size)]
    vector_db = None
```

```
    #遍历每个批次进行向量化处理
        for i, batch in enumerate(all_batches):
            if i == 0:
                vector_db = FAISS.from_documents(
                    documents=batch,embedding=embed_model,
                    distance_strategy="COSINE")
            else: #保存向量数据库到指定路径
        vector_db.save_local("faiss_diabetes_index")
        return vector_db
```

3）检索器实现(new_retriever.py)

检索器的主要作用是根据用户的查询,从向量数据库中找出与之最相似的文本。在这个系统中,使用向量数据库的相似性搜索功能,通过设置合适的搜索类型和参数,能够快速准确地定位到相关的文本信息。

```
#使用 FAISS 向量检索,search_type 为相似性搜索,能找出与查询最相似的文本
retriever=vector_db.as_retriever(
search_type="similarity",
search_kwargs={"k":5}) #k 表示返回最相似的前 k 个结果
    return retriever
```

4）生成模型的配置(generation_model.py)

这部分代码的任务是配置生成模型,用于根据检索到的文本和用户的问题生成详细的回答。

```
#定义要加载的模型,同时加载 tokenizer 用于将文本转换为模型可处理的输入
    model_name = "IDEA-CCNL/Wenzhong-GPT2-110M"
    tokenizer = AutoTokenizer.from_pretrained(model_name,trust_remote_code=True)
    #初始化文本生成管道,设置生成的最大新 token 数、温度、重复惩罚等参数
    medical_pipeline = pipeline(
        "text-generation",model=model,tokenizer=tokenizer,max_new_tokens=256,
        temperature=0.3,repetition_penalty=1.1,
        pad_token_id=tokenizer.eos_token_id)
    # 将生成管道集成到 LangChain 中
    medical_llm = HuggingFacePipeline(pipeline=medical_pipeline)
    return medical_llm
```

5）主程序

主程序是整个医疗问答系统的核心调度部分,它会依次调用前面实现的各个功能模块,完成系统的初始化和示例问题的处理。具体流程如下：首先,进行知识库的准备工作,将PDF 文档加载并分割成文本块；接着,构建向量数据库,把文本转换为向量形式以便快速检索；然后,配置检索器,用于根据问题从向量数据库中查找相关文本；再配置生成模型,用于根据检索结果生成回答；最后,构建 RAG 链(检索增强生成链),将检索和生成过程结合起来,对示例问题进行处理并输出答案和来源文档。

```
#获取文档内容所在目录
    directory = os.path.dirname(os.path.abspath(__file__))
    file_path = os.path.join(directory, "Diabetes1.pdf")
#调用知识库准备函数,将 PDF 文档加载并分割成文本块
    texts = prepare_knowledge_base(file_path)
```

```
# 调用函数构建向量数据库,将文本转换为向量形式
    vector_db = build_vector_database(texts)
# 从向量数据库创建检索器,用于后续查询相关文本
    retriever = vector_db.as_retriever(
        search_type="similarity",search_kwargs={"k": 5})
# 配置生成模型,用于根据检索结果生成回答
    llm = configure_generation_model()
# 构建 RAG 链,将检索和生成过程结合起来
    rag_chain = (
            {"context": retriever | format_docs, "question": RunnablePassthrough()}
            | PROMPT| llm| StrOutputParser())
    # 定义示例问题并调用 RAG 链处理问题并得到答案
    question = "我国糖尿病患病率的变化趋势是怎样的?"
    result = rag_chain.invoke(question)
    print(result)
    # 调用检索器获取来源文档
    docs = retriever.invoke(question)
```

运行结果如图 5.2 所示。

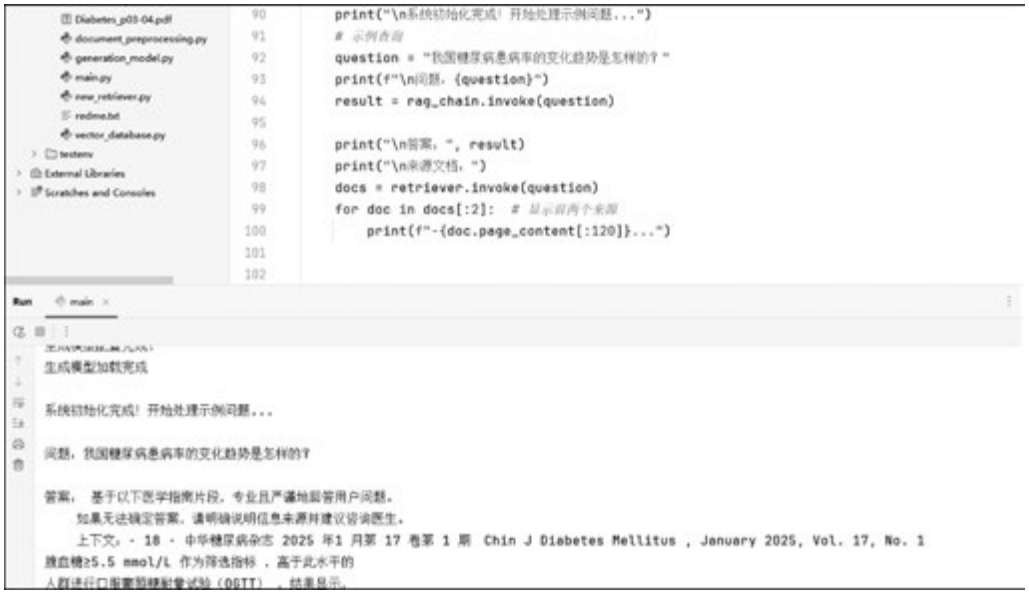


图 5.2 运行结果

5.5 实验:基于 LangChain 的医学领域 RAG 系统实现

1. 实验意义与目的

以 LangChain 框架为核心,一步一步教读者实现一个完整的 RAG 系统,覆盖环境配置、代码解析。通过医疗问答系统的实战案例,更深入地理解 RAG 系统。

2. 实验内容

(1) 完成 RAG 环境的搭建。

(2) 完成 RAG 系统示例。

(3) 启动 RAG 系统。

(4) 与 RAG 交互。

3. 实验环境

(1) 操作系统：Windows 10/11 64 位。

(2) 硬件。

最低：12GB 内存。

推荐：32GB 内存，NVIDIA CUDA 显卡（RTX 3060+推荐）。

存储：50GB 硬盘空间（推荐 SSD）。

软件：Pycharm、Anaconda。

4. 实验步骤

1) 安装软件

(1) 安装 Pycharm。

访问 Pycharm 官网(https://www.jetbrains.com.cn/pycharm/download/?section=windows)，下载安装包，如图 5.3 所示。

然后运行 pycharm-2025.1.exe，进行 Pycharm 安装，均选默认配置安装即可。

安装完成后，运行桌面图标，如图 5.4 所示。

图 5.3 下载 Pycharm 界面

图 5.4 Pycharm 软件图标

(2) 安装 Anaconda。

访问 Anaconda 官网(https://www.anaconda.com/download/success)，单击 Download 按钮，下载安装包，然后运行 Anaconda3-2024.10-1-Windows-x86_64.exe，如图 5.5 所示。

然后运行 Anaconda3-2024.10-1-Windows-x86_64.exe，进行 Anaconda 安装，均选默认配置安装即可。

安装完成后，运行桌面图标，如图 5.6 所示。

2) 搭建 Python 环境

(1) 使用教材提供的已建立好的 Python 环境(推荐)。

① 下载完毕教材提供的文件资源，并找到 rag_env.zip，并解压放置到本地路径中(本环境基于 CPU 版本)，如图 5.7 所示。

② 找到文件资源中“代码”文件夹中的“第五章 代码”文件，并解压放置到本地路径中，如图 5.8 所示。

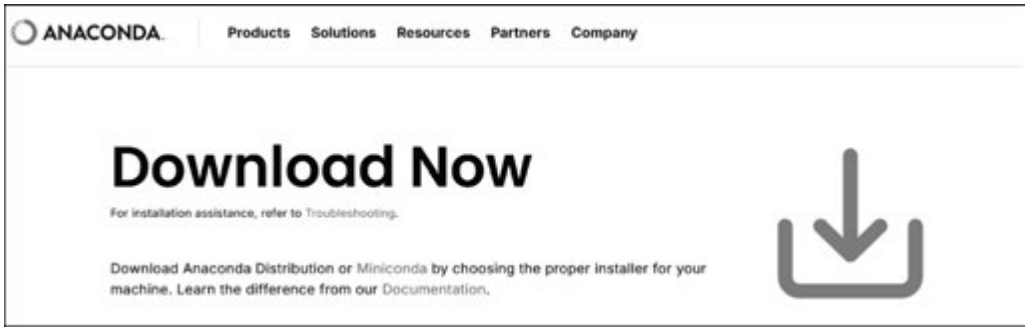


图 5.5　下载 Anaconda 界面

图 5.6　Anaconda 软件图标

图 5.7　Python 环境

图 5.8　第五章内容的代码

③ 打开 Pycharm，选择打开项目，找到刚解压的代码文件夹，如图 5.9 所示。

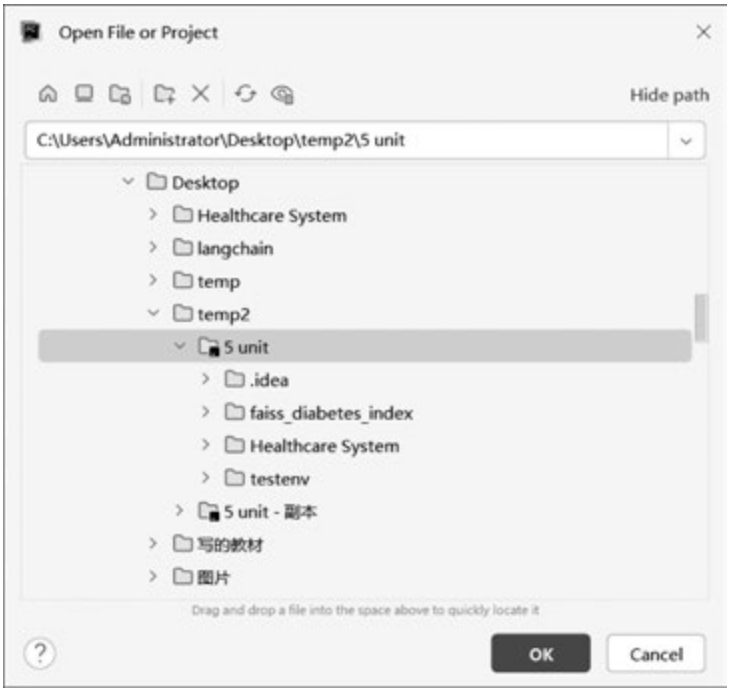


图 5.9　使用 Pycharm 打开第五章代码

（2）自主安装 Python 环境。

这里借助 Anaconda 工具来高效管理 Python 环境，将使用 CPU 版本进行开发，确保程序的稳定运行。

① Python 环境配置：建议 Python 3.9 或以上版本。

conda create -n rag_env python=3.9

```
(base) C:\Users\Administrator>conda create -n rag_env python=3.9
```

conda activate rag_env

```
(base) C:\Users\Administrator>conda activate rag_env
(rag_env) C:\Users\Administrator>_
```

② 核心依赖与安装命令。

faiss-cpu：faiss 库能高效构建和管理向量数据库。

```
(rag_env) C:\Users\Administrator>conda install -c conda-forge faiss-cpu=1.7.4
```

sentence-transformers：用于生成文本嵌入，HuggingFaceEmbeddings 依赖此库。

huggingface_hub：提供模型下载与加载、数据集等功能。

```
(rag_env) C:\Users\Administrator>conda install -c conda-forge sentence-transformers huggingface_hub
```

langchain：为构建 RAG 系统提供所需的组件。

langchain-community：community 库主要包含文档加载器、嵌入模型等工具。

langsmith：是与 LangChain 应用平台进行交互的 Python 客户端库，提供执行追踪、错误调试、评估等功能。

```
(rag_env) PS C:\Users\Administrator> conda install -c conda-forge langchain langchain-community langsmith
```

Pypdf：用于处理 PDF 文件的 Python 库。

```
(rag_env) C:\Users\Administrator>conda install -c conda-forge pypdf
```

③ 环境验证：项目资源中提供两个测试脚本，test_rag_env.py、test_rag_env2.py 可自行测试。

测试程序 1：test_rag_env，如图 5.10 所示。

图 5.10 test_rag_env 测试成功

测试程序 2：test_rag_env2，如图 5.11 所示。

3）启动程序

（1）执行 main.py 脚本，如图 5.12 和图 5.13 所示。

（2）执行成功，如图 5.14 所示。

图 5.11　test_rag_env2 测试成功

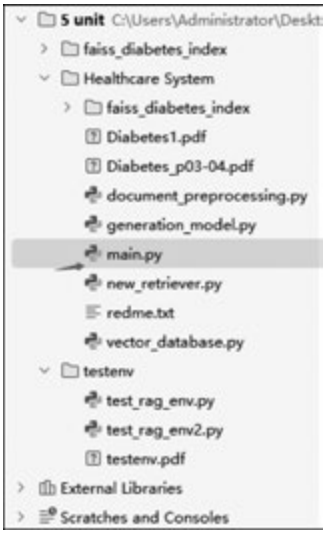


图 5.12　main.py 文件位置

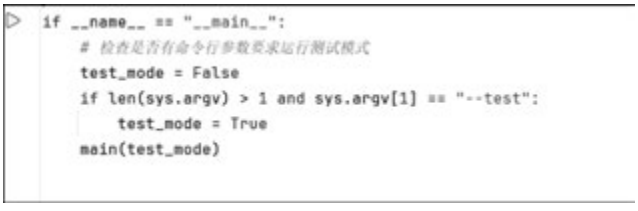


图 5.13　执行 main 脚本

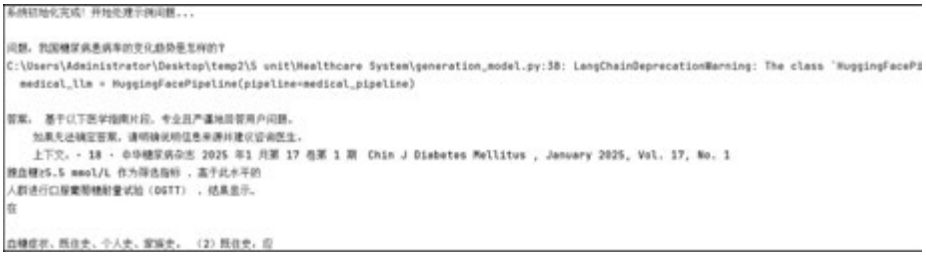


图 5.14　执行成功

（3）更换寻求 RAG 系统的问题：找到 main.py 第 92 行代码，可以输入自己想要答案的问题，如图 5.15 所示。

注意：

① 执行过程中根据用户自身的计算机性能，可以更换不同模型：找到 generation_model.py，找到 model_name 行，可以自行更换本书中提供的其他预训练模型，如图 5.16 所示。

```
question = "我国糖尿病患病率的变化趋势是怎样的？"
print(f"\n问题：{question}")
result = rag_chain.invoke(question)

print("\n答案：", result)
print("\n来源文档：")
docs = retriever.invoke(question)
for doc in docs[:2]:  # 显示前两个来源
    print(f"-{doc.page_content[:120]}...")
```

图 5.15　更换咨询问题

图 5.16　预训练模型的匹配

② 本书中代码所采用的模型为 IDEA-CCNL/Wenzhong-GPT2-110M，由于该模型是小容量预训练中文大模型，因此，该模型可加载的 PDF 文件内容较少，示例代码中仅使用了 Diabetes1.pdf 数据，该数据仅截取原数据集部分内容，若要使用完成数据集，可找到“数据集”文件夹中的“中国糖尿病防治指南”，同时请更换其他大模型，否则会报容量不足错误，如图 5.17 所示。

图 5.17　运行结果影响的文件

5. 实验结果与分析

本地部署 RAG 系统成功，实现回答更精准，快速响应，同时也注意数据隐私保护。

小　结

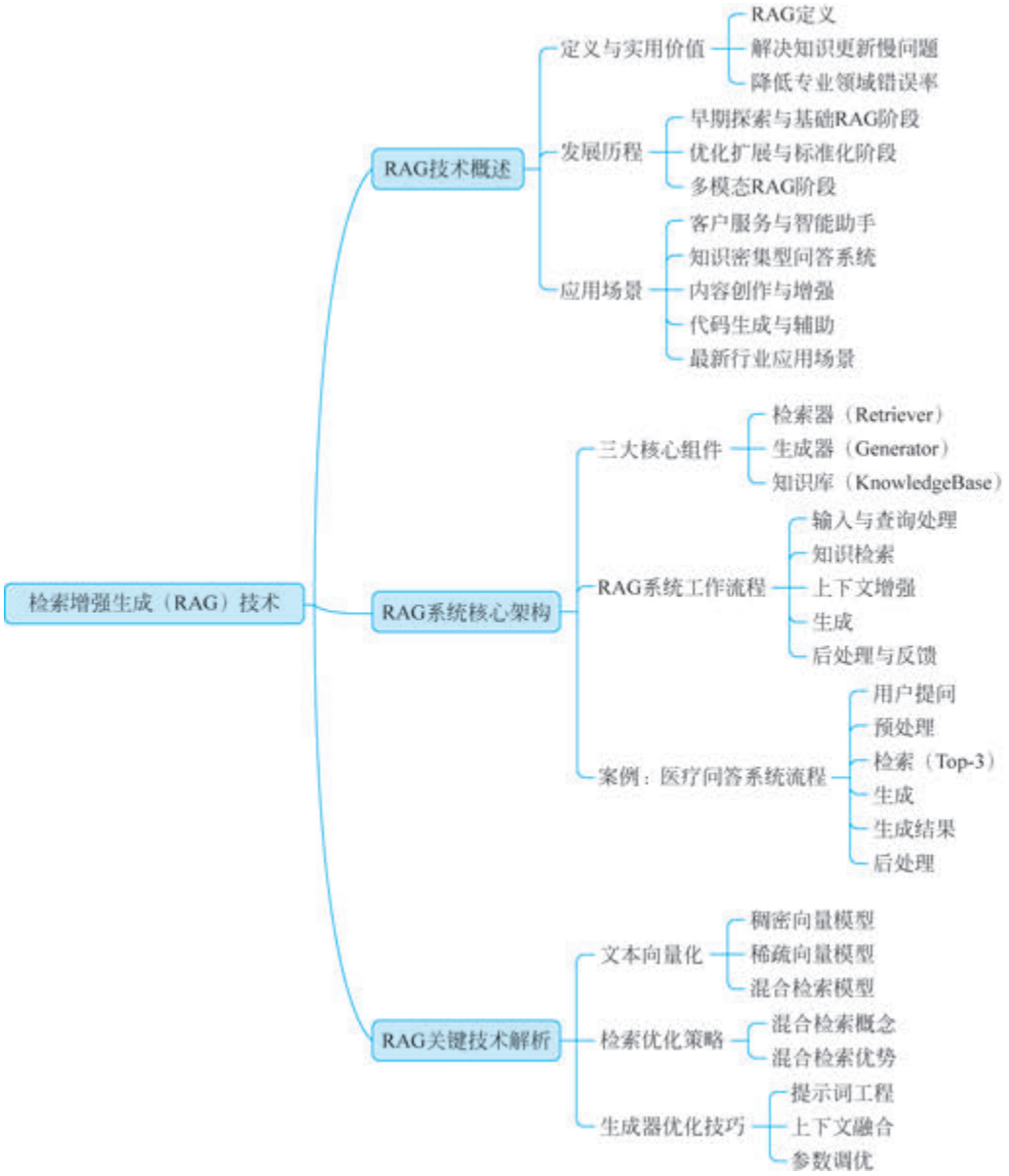
检索增强生成(RAG)技术通过结合信息检索与文本生成，有效提升了大语言模型在知识更新、专业领域应用及生成内容可靠性等方面的能力。RAG 技术的发展历程经历了从早

期探索到多模态融合的演进，并在智能客服、医疗保健、教育培训等多个行业展现出应用潜力。

RAG 系统的核心架构包含检索器、生成器和知识库三大组件，通过“检索-生成”的工作流程，实现从用户查询到答案输出的完整链路，关键技术包括文本向量化、混合检索及提示词工程等，这些技术共同作用于提升 RAG 系统性能和优化生成结果。

总体而言，RAG 技术通过解决大语言模型的技术瓶颈，实现了更高效、准确和可靠的文本生成，为人工智能在各行业的深化应用提供了有力支持。

思维导图



科学家传奇：颜水成与RAG技术的“破壁”之旅

在人工智能(AI)这片充满无限可能的星空中，一颗耀眼的新星正在冉冉升起，他的名字叫颜水成。他就像一位无畏的探险家，在AI的莽莽丛林中，披荆斩棘，勇往直前，而他与一项被称为RAG(检索增强生成)的革命性技术之间的故事，更是一段激动人心的传奇。

好奇心：一切开始的起点

颜水成的好奇心就像一颗埋藏在他心底的种子，从小就生根发芽。转动的风车在他眼里是空气动力学的奥秘，滴答的钟表在他耳中是时间流逝的神秘低语。这种对世界的好奇，最终指引他走上了探索科学的道路。

AI的召唤：燃烧的梦想

随着颜水成渐渐长大，他对科学的热爱也如同烈火般燃烧起来。在众多学科中，计算机科学和AI就像一块巨大的磁石，深深地吸引着他。那些精妙的算法和模型，就像一扇扇通往未知世界的大门，让他忍不住想要一探究竟。

大学生活就像一头扎进知识的海洋，颜水成贪婪地吸收着一切。他如饥似渴地学习着AI大模型语言相关的知识，就像夜空中突然亮起的一颗星，照亮了他前进的方向。

AI困境：RAG技术的“灵光乍现”

当时的AI领域正处于高速发展的黄金时代，但也面临着巨大的挑战。如何让机器更好地理解和生成人类语言，成为制约AI发展的关键难题。颜水成敏锐地意识到，RAG技术就像黑暗中的一道光，也许就是解决这个难题的“钥匙”。

RAG技术：AI的“最强大脑”

你是否曾被AI一本正经地胡说八道震惊过？RAG技术正是为了解决AI的这个“幻觉”问题而生。它就像给AI装上了一个“外脑”，让AI在回答问题前，先从外部知识库中“查阅资料”，确保答案的准确性和可靠性。

RAG技术就像AI的“最强大脑”，它有三大“必杀技”。

- **实时更新**：告别AI知识过时的难题，RAG让AI永远掌握最新信息。
- **专业精通**：解决AI在专业领域容易出错的问题，RAG让AI成为真正的专家。
- **有理有据**：改变AI“黑箱”操作的弊端，RAG让AI的思考过程可以追溯。

颜水成的“破壁”之旅

为了实现这个目标，颜水成带领他的团队，踏上了一条充满挑战的道路。他们遇到了无数难以想象的困难，经历了无数次的失败。

在一次关键的实验中，他们遇到了一个巨大的难题：模型生成的文本毫无逻辑，就像一盘散沙。团队成员们尝试了各种方法，却都毫无进展。

颜水成陷入了沉思。他不断地回顾实验的每一个细节，突然，一个全新的想法闪现在他的脑海中：调整数据检索策略！他认为，如果能设计出更智能的数据检索方式，也许就能为模型注入清晰的逻辑。

这个想法点燃了团队新的希望。他们夜以继日地工作，最终设计出了一种全新的数据检索算法。实验结果出来了！模型生成的文本终于变得连贯而有逻辑，就像一个真正的人在思考和表达。

RAG 技术的未来：无限可能

RAG 技术的成功为人工智能开辟了新的道路。它正在各行各业展现出巨大的应用潜力。

- **智能客服：**RAG 让 AI 客服秒懂用户问题，提供精准解答。
- **法律咨询：**RAG 帮助律师快速查找法律条文，提高工作效率。
- **新闻写作：**RAG 辅助记者撰写深度报道，让新闻更有价值。
- **代码生成：**RAG 帮助程序员快速解决 Bug，提升开发效率。

而颜水成，这位 AI“最强大脑”的缔造者，仍在继续他的探索，带领我们走向更加激动人心的 AI 未来。

习 题 5

扫一扫

习题

扫一扫

自测题