查询结构化不仅仅是将自然语言转换为结构化查询语言，还需要结合语义信息和元数据，以构建更复杂和准确的查询。通过将用户意图与数据结构相结合，系统能够生成更强大的查询语句。例如，Text-to-SQL 技术能够将自然语言问题转换为 SQL 语句，从关系型数据库中提取答案；Text-to-Cypher 则用于处理图数据查询，基于图结构返回更精确的结果。这种方式使 RAG 系统能够在融合多种数据类型的同时，确保查询的精准性和多样性，从而提供更全面的答案和更优质的用户体验。

9.2.3 检索

检索模块在 RAG 系统中扮演着至关重要的角色。在 RAG 系统中，检索模块需要能够高效地处理大量的文本数据，并且需要能够准确地识别和匹配查询和文档之间的语义相似性。因此，检索模型的选择和优化对于 RAG 系统的性能至关重要，因为它们直接影响到检索的准确性和效率。检索模型还需要能够适应不同的数据类型和查询类型，以确保在各种场景下都能够提供准确的检索结果。目前的检索主要分为：稀疏检索、稠密检索和混合检索。本节将分别介绍上述检索模型。

1. 稀疏检索

稀疏检索（Sparse Retrieval）是一种基于统计特征的方法，通过将查询和文档转换为稀疏向量来实现检索。稀疏向量的特点是大部分元素为零，仅保留少量非零值，这使得计算更加高效且存储成本较低。许多经典的信息检索方法，如 TF-IDF 和 BM25，都是稀疏检索的典型实现。这些方法通过词频、逆文档频率等显性统计特征对查询和文档进行建模，能够快速匹配相关内容。稀疏检索架构如图9.7所示。

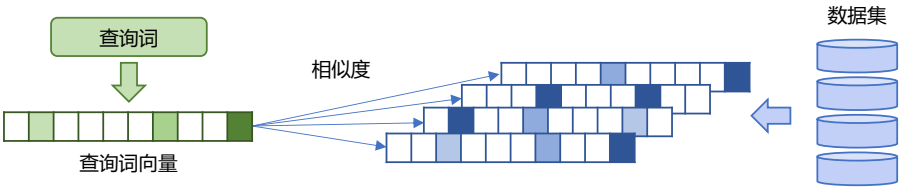

图 9.7 稀疏检索架构图

稀疏检索的最大优势在于其高效性，尤其适用于处理大规模文档库的检索任务。由于稀疏向量中仅计算非零元素的部分，相较于密集向量方法，其计算复杂度显著降低。因此，稀疏检索在资源有限或实时性要求较高的场景中表现尤为突出。稀疏检索器在大规模数据集上的效率使其成为工业界的主流选择之一。

尽管稀疏检索在效率上具有明显优势，但其在捕捉复杂语义关系方面存在局限性。由于稀疏方法主要依赖显性统计特征，如词频和词项匹配，无法有效处理同义词、上下文语义等深层语义

信息。例如，对于“汽车”和“车辆”这样的同义词，稀疏检索器通常无法感知两者的语义相似性，从而可能导致检索结果的相关性下降。稀疏向量的低语义表达能力限制了其在语言理解任务中的适用性。

2. 稠密检索

稠密检索（Dense Retrieval）是一种通过深度学习模型将查询和文档编码为稠密向量（Dense Vectors）的检索方法。与稀疏向量不同，稠密向量的每个维度都可能有值，从而能够捕捉更丰富的语义信息。这种方法依赖预训练语言模型（如 BERT、RoBERTa）或特定的双塔模型（Dual Encoder）来生成语义嵌入，使得查询和文档在语义空间中更接近，从而更好地匹配用户意图。在语义搜索、问答系统和对复杂查询的处理任务中，稠密检索表现出了显著的优势。稠密检索架构如图9.8所示。

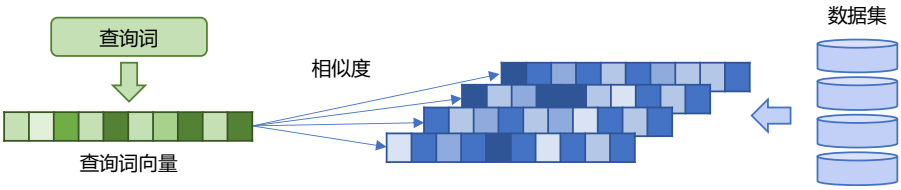

图 9.8　稠密检索架构图

稠密检索的核心优势在于其强大的语义表达能力。由于深度模型能够理解上下文信息和复杂的语义关系，稠密向量不仅可以捕捉显性特征，还能处理同义词、上下文依赖和多层次语义。例如，对于“汽车”和“车辆”这样的同义词，稠密检索器可以识别它们在语义上的相近性，从而提高检索结果的相关性。稠密检索在捕捉细粒度语义关联方面优于传统的稀疏检索方法。

然而，稠密检索也面临一些挑战，特别是在计算成本和存储要求方面。由于稠密向量通常是高维向量（例如 768 维或更高），因此处理和存储大规模文档库的稠密向量需要更高的计算资源。此外，稠密检索依赖于深度学习模型的训练，模型的质量和训练数据的规模直接影响检索效果，这可能增加系统的开发复杂性和维护成本。稠密检索的高计算需求限制了其在资源受限场景中的应用。尽管如此，稠密检索已经成为 RAG 系统和现代信息检索中的重要方法，尤其是在需要高语义理解能力的任务中。

3. 混合检索

混合检索（Hybrid Retrieval）是一种结合稀疏检索和稠密检索优势的检索方法，用于提升检索系统的效率和效果。稀疏检索（如 TF-IDF 和 BM25）擅长处理显性特征，能够快速匹配高频词项，同时在大规模文档库中表现出极高的计算效率；而稠密检索（如基于深度学习模型生成的语义向量）能够捕捉复杂的语义关系，对理解同义词、上下文和深层语义非常出色。混合检索通过将两

者结合，既保留了稀疏检索的高效性，又增强了系统在语义理解上的能力。

混合检索的核心思想是将稀疏向量和稠密向量的得分进行融合，或者在检索流程中分阶段使用两者。例如，在第一阶段，使用稀疏检索从大规模文档库中快速筛选出一个候选集合（通常称为“粗排”）；在第二阶段，对候选文档进行稠密检索或语义重排序，以提升结果的相关性。这种分阶段策略既能降低稠密检索的计算成本，又能显著提高检索质量。混合检索在效率和效果之间达到了良好的平衡。混合检索架构如图9.9所示。

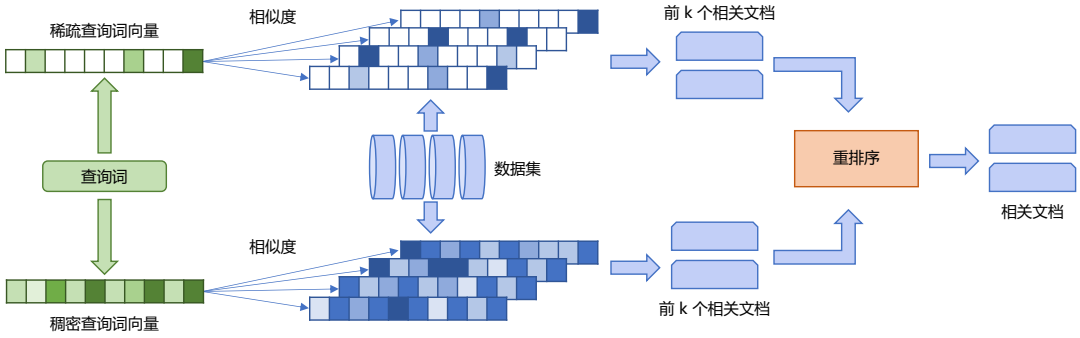

图 9.9 混合检索架构图

混合检索的优势在于其灵活性和适应性。对于需要显性词项匹配的查询（如“精确匹配”类问题），稀疏检索能够快速捕捉关键词；而对于需要语义理解的复杂查询（如自然语言表达的长尾问题），稠密检索能够提供更相关的结果。此外，混合检索可以根据不同的应用场景调整稀疏和稠密部分的权重，从而实现个性化的优化。例如，在工业界的大规模搜索引擎中，混合检索被广泛应用于广告推荐、问答系统等场景，表现出了良好的效果。

尽管混合检索方法在许多场景中表现优异，但其设计和实现也存在一定的技术挑战。首先是如何有效融合稀疏向量和稠密向量的得分，因为两者的分布和尺度不同，需要设计合理的归一化或加权策略。其次，混合检索的计算开销依然较高，尤其是在需要实时处理大规模用户查询时，如何进一步优化效率是一个重要问题。随着硬件性能的提升和检索算法的优化，混合检索有望在未来的信息检索系统中占据更加重要的地位，为用户提供更高效、更精准的检索服务。

9.2.4 检索后优化

检索后优化（Post-retrieval processing）是优化大语言模型生成效果的重要步骤。直接将检索到的文本块输入大语言模型并不能得到最好的结果，存在诸多挑战。首先，大语言模型与人类类似，对长文本往往只能记住开头和结尾部分，而容易遗忘中间内容，这被称为“中间遗忘”（lost in the middle）问题。其次，检索到的文本中可能包含噪声信息或与事实相悖的内容，这些“噪声/反

事实”文本会对最终生成结果产生负面影响。此外，大语言模型的上下文窗口长度有限，即使检索到了大量相关内容，也无法全部纳入模型处理。因此，通过对检索内容进行后处理，可以更好地利用上下文信息，从而提升模型的生成质量和可靠性。

本节将详细介绍后检索模块的常见组成部分，包括重排序（Rerank）、内容压缩以及内容选择等步骤。

1. 重排序

在检索增强生成系统中，重排序（Rerank）是一个关键组件，其主要目的是对检索到的文章片段（chunks）进行重新排序，以提升结果的相关性和多样性。重排序的作用是基于特定的排序算法或模型，将更重要的内容优先呈现，同时避免重要信息被冗余或低相关性的内容掩盖。重排序算法从大类上可以分为基于规则和基于模型两大类。

基于规则的重排序（Rule-based Rerank）是一种常用的重排序方法，通过计算特定的指标对数据块进行排序。常见指标包括多样性（Diversity）、相关性（Relevance）和最大边际相关性（Maximal Marginal Relevance，MMR）[427]。MMR 是一种结合查询相关性和信息新颖性的排序方法，可以有效减少冗余并增强结果的多样性。例如，在选择关键短语时，MMR 会优先考虑与查询高度相关且不重复的短语，从而平衡结果的相关性和信息量。这种基于规则的方式简单高效，适用于许多具有固定规则需求的场景。

基于模型的重排序（Model-based Rerank）则利用语言模型对数据块进行排序，通常通过计算数据块与查询之间的相关性来完成。这种方法能够动态地根据查询上下文判断数据块的重要性，从而生成更精准的排序结果。重排序模型的技术持续迭代，已经从文本数据扩展到多模态数据（如表格和图像），实现了更广泛的应用场景。相比于规则方法，基于模型的重排序能够捕捉更复杂的语义关系，特别是在需要理解深层次上下文的任务中表现突出。因此，重排序在 RAG 系统中不仅是提升检索质量的重要工具，也为多模态数据处理提供了强有力的支持。

2. 内容压缩

将大量相关文档段拼接为冗长的上下文通常会引入噪声，削弱大语言模型对关键信息的感知能力。为解决上述问题，压缩（Compression）方法核心目标是通过内容压缩减少噪声，同时保留信息完整性，以提高语言模型的推理效率。

内容压缩的一种方法是通过小型语言模型（如 GPT-2 Small 或 LLaMA-7B）对检索内容进行对齐和预训练，以检测并移除提示中的不重要信息[428]。这种方法能够大幅度减少输入上下文的冗余内容，将原始输入转化为一种更适合大语言模型理解的形式，而无需对大语言模型进行额外训练。具体而言，通过函数 $f_{\text{comp}}(q, D^q)$，将检索到的文档集合 D^q 压缩为 D_c^q，其中每个文档内容的长度 $|d_i^{qc}|$ 小于原始文档的长度 $|d_i|$。这种方法不仅能保持上下文的语言完整性，还能实现高效的压缩比，使得输出在语义上对模型更有意义，即便对人类而言可能变得难以理解。这种直接的压缩方法在保持性能的同时，简化了实现难度，适用于多种实际场景。

另一种直接而有效的内容压缩方法是利用大语言模型对检索内容进行评估（LLM-Critique）。通过让 LLM 对检索得到的内容进行自我审查，可以过滤掉相关性较差的文档。例如，在 Chatlaw[429] 系统中，构造了评估提示词，大语言模型对参考的法律条款进行自我建议和评估，以判断其与查询的相关性。这种方法能够在生成最终答案前移除低质量或无关内容，从而优化输入上下文的质量。

3. 内容选择

内容选择（Selective Context）是检索增强生成系统中优化输入上下文的重要方法。其核心目标是通过识别和移除冗余信息，保留最为关键的内容，从而提高语言模型的推理效率和结果质量。内容选择的关键在于计算输入内容的自信息量（Self-Information），这是一种衡量内容信息价值的指标。自信息量越高，表明该内容在上下文中越稀有且重要。在实际应用中，基础语言模型对检索到的文档内容逐词评估，删除信息价值较低的部分，仅保留对任务有贡献的高信息量内容。这一过程能够有效精简输入上下文，减少噪声干扰。

这种方法的主要优势在于提升了语言模型的专注性，使其能够更高效地处理长上下文输入，并对关键信息作出准确的推理。同时，这种精炼方法能广泛适用于法律分析、学术文献综述和问答系统等任务场景，而不会显著影响模型性能。然而，内容选择也存在一定局限性。它可能忽略被删除内容之间的相互依赖关系，导致上下文完整性受损。此外，内容选择通常依赖于小型语言模型的计算，而这些模型与目标语言模型可能在理解能力上存在对齐问题，进而影响压缩内容在推理任务中的效果[430]。

9.2.5 生成

生成模块是整个 RAG 系统的的核心模块，负责利用大语言模型结合用户查询与检索到的上下文信息生成答案。生成的内容需要与检索阶段获取的关键信息保持一致，确保知识的整合与输出的准确性。还需要根据用户的指令、场景上下文以及个人偏好对内容进行调整，使其更加符合具体的使用场景和个性化要求。这种对相关知识的整合和对多样化需求的适应，确保了 RAG 系统生成的内容既具有上下文相关性，又能够满足用户的特定需求，从而在实际应用中展现出强大的灵活性和实用性。

例如，用户输入“使用 500 字介绍一下复旦大学的历史沿革”，通过此前预检索、检索以及后检索模块，生成模块输入给语言模型的内容如下所示：

```
<chunk id="1">
复旦大学校名取自《尚书大传》之“日月光华，旦复旦兮”，始创于 1905 年，原名复旦公学，1917 年
定名为复旦大学，是中国人自主创办的第一所高等院校。上海医科大学前身是 1927 年创办的国立第四
中山大学医学院。2000 年，复旦大学与上海医科大学合并。目前，学校拥有哲学、经济学、法学、教
育学、文学、历史学、理学、工学、医学、管理学、艺术学、交叉学科等 12 个学科门类；2021 年，学
校 20 个学科入选第二轮“双一流”建设学科，比首轮增加 3 个入选学科。
</chunk>

<chunk id="2">
肇始吴淞（1905—1911，校址：吴淞）
1902 年，马相伯倾其家产，借天主教徐家汇天文台余屋为校舍，创办震旦学院。1905 年，为反抗教会
势力干预校政，于右任、邵力子等 130 名学生愤然脱离震旦，支持马相伯在吴淞复校。1905 年 9 月 14 日
(阴历八月十六)，国人自办的第一所高等学校——复旦公学在上海吴淞提督行辕正式开学。
</chunk>

<chunk id="3">
创校吴淞（1927—1931，第四中山大学医学院，校址：吴淞）
1927 年，由中国人自主创办的第一所国立医学院——国立第四中山大学医学院（上医前身）在上海吴淞
建立。创始人颜福庆、乐文照、高镜朗等始终秉持着“为人群服务，为人群灭除病苦”的朴素信念，并
融注于医学教育和医学实践的日常。
</chunk>

<chunk id="4">
强强联合（2000—2010，复旦大学，校址：邯郸路、枫林路、淞沪路、张衡路）
2000 年 4 月 27 日，复旦大学与上海医科大学强强联合，组建新的复旦大学。复旦发展成为文理医三足
鼎立，在国内外享有盛誉的综合性研究型大学。2005 年，复旦大学隆重庆祝建校一百周年，进一步明
确了建设具有世界一流水平的社会主义综合性大学的目标。探索贯通本科教育全过程的通识教育新模
式，打造以培养探究能力为核心的拔尖创新人才培养体系。校地扩展为邯郸、枫林、江湾、张江四校区。
</chunk>


instruction:
使用 500 字介绍一下复旦大学的历史沿革
```

通过上例可以看到，RAG 系统在进行问题回答时，不再完全依赖模型内生的记忆，而是通过检索外部知识库来生成更准确和丰富的回答。很多知识细节不再需要模型准确记忆。这种机制充分利用了检索和生成的结合优势，在面对知识更新快、领域复杂或模型训练数据中未覆盖的信息

时，显得尤为重要。相比传统的语言模型，RAG 系统通过检索阶段获取最新或特定领域的信息，克服了模型内生记忆的局限性，尤其是在处理长尾问题或细分领域的专业知识时，可以表现更加出色。

9.2.6 编排

编排模块是 RAG 系统中的核心控制单元，它负责在关键节点进行决策并动态选择后续步骤。与传统固定流程的僵化方法不同，编排模块引入了灵活的适应能力，可以根据先前结果实时调整流程。这种模块化、动态化的特性是 Modular RAG 的标志性特点，展现出更高的智能化和灵活性。本节将分别介绍编排模块的主要模块，包含路由（Routing）、调度（Scheduling）以及融合（Fusion）。

1. 路由

在响应多样化查询的过程中，RAG 系统可以通过路由机制将查询分配到针对不同场景设计的特定管道中。这种机制是一个通用性较强的 RAG 架构的重要特性，能够处理各种复杂的情境需求。路由模式可以分为三种主要类型：元数据路由、语义路由以及混合路由。

元数据路由（Metadata Routing）基于查询中提取的关键术语或实体，通过与预设关键词集合的匹配来优化路由流程。每个 RAG 流程都定义了一组关键词，当查询中的关键词与某流程的关键词集合匹配度较高时，该流程就会被选为处理流程。匹配分数由关键词的重叠比例计算得出。元数据路由适合对显性关键词高度敏感的场景。整个过程可以形式的化表示为，对于特定的检索增强生成（RAG）流程，记为 F_i，预先定义的路由关键词表示为集合 $K_i = \{k_{i1}, k_{i2}, \ldots, k_{in}\}$。在查询 q_i 中识别出的关键词被指定为 K_i'。查询 q 的匹配过程通过关键得分方程来量化：

$$\text{score}_{\text{key}}(q_i, F_j) = \frac{1}{|K_j'|}|K_i \cap K_j'| \tag{9.2}$$

该方程计算预先定义的关键词与在查询中识别出的那些关键词之间的重叠部分，并通过 K_j' 中关键词的数量进行归一化。最后一步是确定与查询 q 最相关的流程：

$$F_i(q) = \underset{F_j \in F}{\text{argmax}}\ \text{score}(q, F_j) \tag{9.3}$$

语义路由（Semantic Routing）则依赖查询的语义信息，通过语言模型计算查询与预定义意图的匹配概率。每个意图对应一个具体的 RAG 流程，路由机制会根据最大匹配概率选择最相关的流程。语义路由更适合需要深层次意图理解的复杂场景，能够捕捉查询中隐含的语义信息。整个过程可以形式地化表示为，给定一个预先定义的意图集合 $\Theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$，查询 q 具有某种意图的概率为：

$$P_\Theta(\theta|q) = \frac{\mathbf{e}^{P_{\text{LM}}(\theta|q)}}{\sum_{\theta \in \Theta} \mathbf{e}^{P_{\text{LM}}(\theta|q)}} \tag{9.4}$$

路由到特定的 RAG 流程由语义得分确定：

$$\text{score}_{\text{semantic}}(q, F_j) = \underset{\theta_j \in \Theta}{\text{argmax}} P(\Theta) \tag{9.5}$$

函数 $\delta(\cdot)$ 充当一个映射函数，它将一个意图分配给一个不同的 RAG 流程 $F_i = \delta(\theta_i)$。

混合路由（Hybrid Routing）结合了元数据路由和语义路由的优点。通过引入权重因子，混合路由在元数据匹配和语义分析之间找到平衡点，从而实现更精确的路由选择。这种方法既考虑显性关键词的匹配，也兼顾深层次语义信息的理解，非常适合在复杂、多样化的查询环境中使用。混合路由可以通过整合语义分析和基于元数据的方法来实现，其定义如下：

$$\alpha_i = \alpha \cdot \text{score}_{\text{key}}(q, F_j) + (1 - \alpha) \cdot \max_{\theta_j \in \Theta} \text{score}_{\text{semantic}}(q, F_j) \tag{9.6}$$

其中 α 是一个权重因子，用于平衡基于关键词的得分和语义得分的贡献。

2. 调度

随着 RAG 系统在复杂性和适应性方面的不断提升，调度（Scheduling）模块主要扮演越来越重要作用，它能够识别关键节点，负责管理和协调系统的各个流程。包括何时需要进行外部数据检索、如何评估生成结果的充分性，以及在必要时决定是否启动进一步的检索。这一模块特别适用于递归、迭代和自适应检索的场景，确保系统能够根据当前任务的需求动态调整流程，从而在适当的时机停止生成或启动新的检索循环。这种智能调度机制使 RAG 系统更高效、更精准地处理复杂任务。调度模型主要三种实现方式，包括规则判断、大语言模型判断以及知识引导调度。

规则判定（Rule Judge）是一个重要的机制，用于评估生成答案的质量并决定进一步的操作。系统通过评分机制对生成的答案进行质量评估，并根据预设的阈值判断是否继续或终止生成过程。具体来说，系统会检查生成答案中每个词的概率是否高于设定的阈值 τ，若满足条件，则接受当前答案；否则，系统会重新生成新答案。这种方法确保了生成内容的可靠性和准确性，同时为系统的迭代改进提供了依据。规则调度可以如下形式化定义：

$$y_t = \begin{cases} \hat{s}_t & \text{如果 } \hat{s}_t \text{ 的所有词元的概率都} \geqslant \tau \\ s_t = \text{LM}([D_{q_t}, x, y_{<t}]) & \text{其他情况} \end{cases} \tag{9.7}$$

其中，$\hat{s}_t$ 表示临时答案，s_t 是语言模型的输出。接受 $\hat{s}_t$ 的条件是其内部的所有词元都必须具有大于或等于阈值 τ 的关联概率。如果不满足这一条件，系统就会转而生成新的答案。

RAG 系统还可以通过大语言模型直接进行判断（LLM Judge）。这一方式包括两种主要方法：第一种方法利用 LLM 的上下文学习能力，通过精心设计的提示来进行决策。这种方法的优势在于无需对模型进行额外的微调，但其判断结果的准确性通常依赖于 LLM 对提示的理解程度。第二种方法通过对 LLM 进行微调，使其生成特定的触发标记，来直接控制模型的行为。例如，借助

Toolformer[389] 的技术构建的 Slef-RAG[431] 方法，可以实现更高的动作响应性，这种方法虽能提升控制精度，但需要大量高质量的指令集对模型进行微调。

知识引导调度（Knowledge-Guided Scheduling）则是一种介于规则判定和完全依赖 LLM 之间的中间方法，通过知识图谱引导信息检索与生成过程[432]。具体来说，系统从知识图谱中提取与问题相关的信息，并构建推理链，将问题拆解为一系列逻辑互联的节点。每个节点包含解决问题所需的关键信息，并据此分别进行信息检索和内容生成。通过这种方式，不仅提高了问题解决的效率和准确性，还使生成的答案具备更清晰的逻辑性和解释力，为复杂问题提供更具条理性的解决方案。

3. 融合

随着 RAG 系统从线性流程发展为复杂的多管道结构，融合（Fusion）模块在其中扮演了至关重要的角色。当系统拓宽检索范围或探索多条管道以提升生成内容的多样性时，融合模块负责高效整合各分支生成的信息。它不仅实现答案的合并，还对内容进行筛选与优化，确保最终输出既全面丰富，又能准确反映问题的多维特性。融合模块的引入，使系统在应对复杂查询时能够提供更加综合且连贯的回答，大幅提升了整体的适应能力与输出质量。融合模块主要包含大语言模型融合、加权继承以及倒数排名融合等方法。

大语言模型融合是多分支信息整合的直接方法之一，利用大语言模型强大的分析与整合能力，将不同分支的信息进行统一处理。然而，这种方法面临一些挑战，特别是在处理超出大语言模型上下文窗口限制的长答案时。为了缓解这一问题，通常会先对每个分支的答案进行摘要提取，提炼关键内容后再输入 LLM，从而在长度限制内保留最重要的信息。这种方法确保了答案的完整性与精确性，即使在处理复杂的多分支生成时也能提供高质量的整合结果。

加权集成是一种基于多分支生成结果的加权选择方法，通过不同分支生成的词元（token）的加权值来综合选择最终输出。具体而言，权重是通过文档与输入查询的相似度得分计算的，使用 Softmax 函数对权重进行归一化，确保所有权重之和为 1。该方法可按如下公式计算：

$$p(y|q, D_q) = \sum_{d \in D_q} p(y|d, q) \cdot \lambda(d, q) \tag{9.8}$$

权重 $\lambda(d, q)$ 由文档 d 和输入查询 q 之间的相似度得分确定。该权重使用 softmax 函数来计算，以确保权重经过归一化且总和为 1。

$$\lambda(d, q) = \frac{e^{s(d,q)}}{\sum_{d \in D_q} e^{s(d,q)}} \tag{9.9}$$

倒数排名融合（Reciprocal Rank Fusion，RRF）是一种集成技术，专门用于将多个检索结果的排名整合为统一的列表。它通过一种定制的加权平均方法，增强了整体预测性能与排名精度[433]。RRF 的核心优势在于其动态的权重分配机制，基于分支之间的相互作用进行调整，特别适合处理

模型或来源异构的场景。在这些复杂情况下，RRF 能够显著提升预测的准确性和整合效果，成为多分支融合的重要工具。

9.3 RAG 系统设计模式

基于 Modular RAG 的设计，各种模式通过模块化操作符之间的协作形成了模块的工作流，称为 RAG 流（RAG flow）。RAG 流可以被分解为由子函数组成的图结构，通过控制逻辑，这些操作符可以按照预定的管道线执行，同时在必要时支持条件判断、分支或循环操作。通过深入分析现有的 RAG 方法，这些模式的模块化特性使其能够灵活适应多样化的场景需求，同时提高了 RAG 系统的设计效率和扩展性。

本章将介绍典型的 RAG 系统模式，包括线性模式、条件模式、分支模式、循环模式等。

9.3.1 线性模式

在 RAG 系统中，线性模式是最简单且最常用的工作流模式，其流程可以分为几个核心模块，包括预检索（Pre-Retrieval）、检索、后检索（Post-Retrieval）以及生成模块，如图9.10所示。当预检索和检索后处理模块缺失时，线性模式会简化为朴素检索增强生成（Naive RAG）范式，仅包含基本的检索和生成过程。常见的线性 RAG 流通过在预检索阶段引入查询变换模块（比如重写或隐式文档扩展（HyDE）操作符），以及在检索后阶段使用排序模块来优化检索结果，从而提升最终生成的质量。

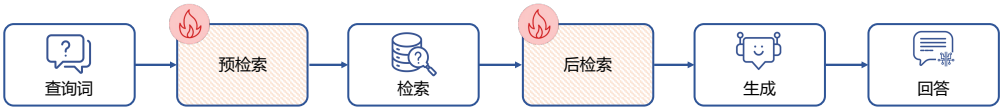

图 9.10　RAG flow 的线性模式[421]

文献 [425] 提出的“重写-检索-阅读”（Rewrite-Retrieve-Read，RRR）方法就是一个典型的线性 RAG 流模式。在预检索阶段，RRR 方法引入了查询重写模块，该模块是基于 T5-large 模型微调的小型可训练语言模型。该模块通过强化学习框架进行优化，将查询重写过程建模为一个马尔可夫决策过程（Markov Decision Process, MDP）。在这一过程中，查询重写模块以大语言模型的最终输出质量作为奖励信号，以此调整和优化生成的查询。具体而言，强化学习通过策略梯度方法对重写模块进行训练，使其生成的查询更符合检索任务的需求，提高检索和生成的整体效率和效果。在检索阶段，RRR 方法使用稀疏编码模型（如 BM25）作为检索工具，从外部知识库中获取与重写后的查询高度相关的文档上下文。

9.3.2 条件模式

条件模式是一种灵活的 RAG 流模式，其核心特点是在不同条件下选择不同的 RAG 流水线，从而针对特定场景进行优化。具体来说，条件模式通过一个路由模块（Routing Module）实现模块的动态选择，该模块根据输入问题的性质决定接下来的流程, 如图9.11所示。例如，面对不同类型的问题，如涉及严肃议题、政治话题或娱乐内容的问题，系统会根据预设条件切换到不同的处理流程。这样的动态路由机制可以显著提升系统对多样化任务的适应能力。

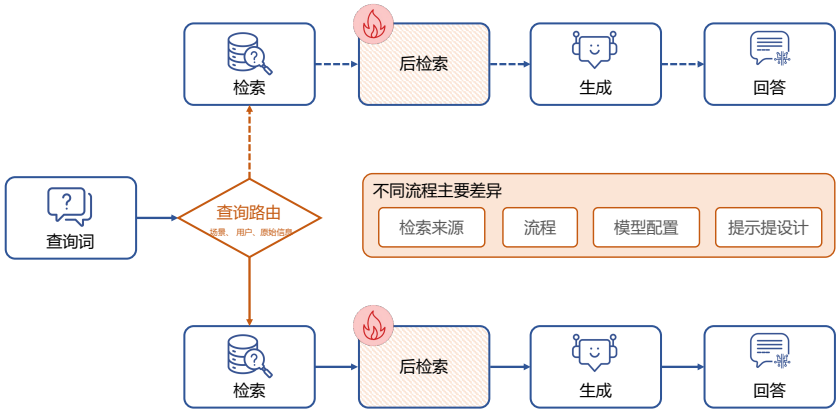

图 9.11　RAG flow 的条件模式[421]

条件模式的分支流通常在以下几个方面存在差异：检索来源、流程、模型配置以及提示设计。例如，对于严肃性较高的问题，系统可能会选择更加可靠的检索来源和严格的生成约束，而对于娱乐类的问题，则可能允许生成更具创意性和娱乐性的回答。通过这种方式，条件模式能够根据任务需求调整 RAG 的各个组件，确保生成的回答既符合场景需求，又具有高相关性和准确性。这种灵活性使得条件模式在处理多样化、复杂性高的任务时具有显著优势。

9.3.3 分支模式

分支模式通过并行运行多个分支的方式增加结果的多样性和鲁棒性。具体来说，分支模式在某个模块中生成多个并行分支，每个分支可以独立执行相同或不同的 RAG 流程。这些流程由多个处理模块组成，生成各自的分支输出结果。随后，所有分支的结果通过聚合函数合并为中间输出结果。重要的是，聚合后的结果并不一定标志着流程的结束，还可以继续传递到后续模块（如验证模块）进行进一步处理。因此，分支模式的整体流程可以表示为从分支生成、独立处理、结果聚合到后续处理的完整流水线。

与条件模式不同，分支模式的特点在于同时运行多个并行分支，而非从多个选项中选择一个分支。分支模式可以根据不同任务需求设计为多种结构类型，通常分为两类：预检索分支模式是

分支间执行不同的流程，以应对复杂场景的多样化需求，如图9.12所示；后检索分支模式是分支间执行相同的 RAG 流程，用于生成多样化的结果，如图9.13所示。通过这样的结构，分支模式能够从多个角度生成和整合信息，从而提升系统的生成能力与结果质量，对多任务处理和复杂场景具有显著优势。

预检索分支（Pre-Retrieval Branching）是一种通过生成多个子查询并并行检索的模式，用于提高检索的全面性和生成结果的多样性。具体而言，该模式从一个初始查询开始，通过查询扩展模块将其扩展为多个子查询。每个子查询随后通过检索模块检索相关文档，形成文档集合。这些文档集合连同对应的子查询一起送入生成模块，生成答案集合。最终，这些生成的答案通过融合模块进行整合，形成最终结果。这种模式通过并行检索与生成，能够从多个角度充分挖掘潜在信息，从而提升生成结果的覆盖度和准确性。

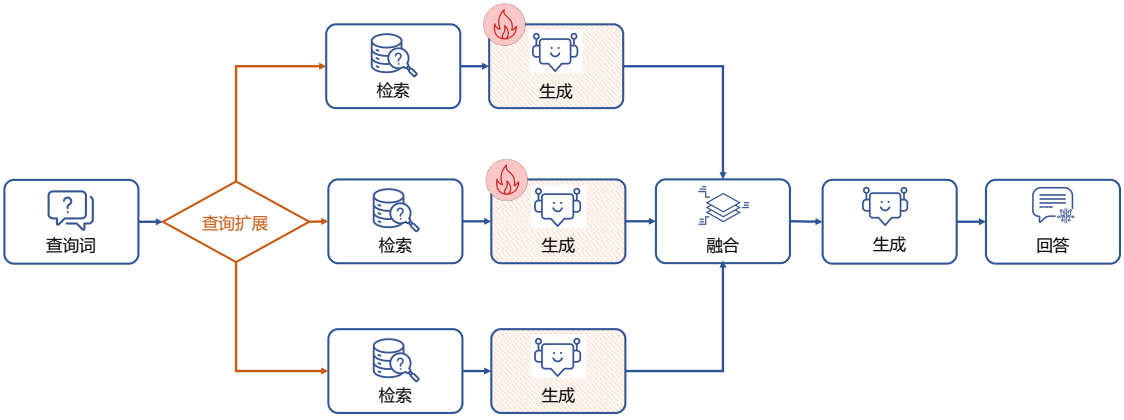

图 9.12　RAG flow 的预检索分支模式[421]

后检索分支（Post-Retrieval Branching）模式则从单一查询开始，通过检索模块获取多个文档块。每个文档块被独立送入生成模块进行处理，生成对应的结果集合。随后，这些生成的结果通过合并模块进行整合，形成最终结果。与预检索分支不同，后检索分支的特点在于单一查询驱动的检索过程，而并行生成则聚焦于对不同文档块的独立处理。该模式适合需要从同一查询结果中挖掘多角度信息的场景，能够充分利用检索到的内容，提高生成结果的多样性和质量。

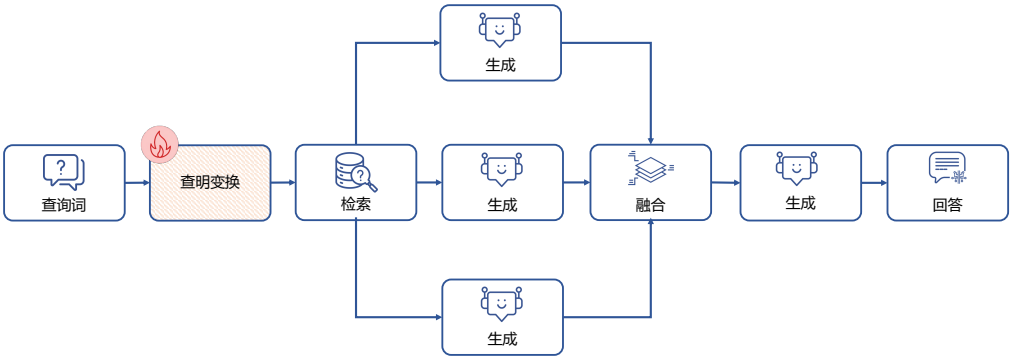

图 9.13 RAG flow 的后检索分支模式[421]

9.3.4 循环模式

循环模式的核心是检索与生成步骤之间的相互依赖性。循环模式通过引入调度模块进行控制，确保系统可以根据需要在特定模块之间重复执行某些操作。这一模式可以被抽象为一个有向图，其中节点代表系统的各个模块，边表示模块之间的控制流或数据流。当一个模块能够返回到之前的模块时，该系统就形成了一个循环结构。这种循环设计允许系统在流程中对某些步骤进行重复优化，从而提升任务的完成效果。

循环模式的关键在于判断模块（Judge Module），用于决定流程是否需要返回到之前的模块或继续向下执行。例如，当一个模块完成后，判断模块可以决定是进入下一个模块还是返回到前置模块。如果系统决定返回，则执行循环操作；如果系统决定不返回，则流程继续向前。这种灵活的控制机制使得循环模式能够动态调整整个流程，从而提高系统的适应性、灵活性以及对复杂任务的处理能力。

循环模式可以进一步细分为三种类型：迭代型、递归型和自适应型（主动型）检索模式。

(1)迭代型循环模式通过多次循环执行检索和生成操作,在每次迭代中逐步优化结果。如图9.14所示，在每一步迭代中，系统根据当前查询和之前的输出结果，检索相关的文档片段，然后利用这些文档生成新的输出。迭代过程通常设置一个最大迭代次数的限制，以避免无限循环。同时，通过判断模块，系统会根据当前生成的结果、历史输出、查询以及检索到的文档来决定是否继续迭代。这种方法能够动态调整检索与生成的过程，逐步获取必要的信息，从而更好地回答复杂问题。

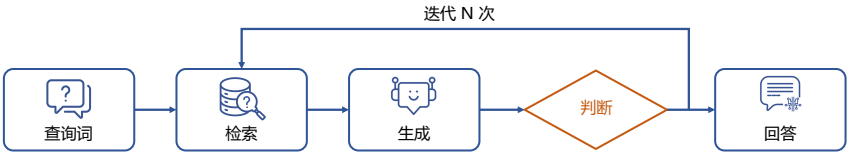

图 9.14　RAG flow 的迭代型循环模式[421]

（2）递归型循环检索模式是一种具有明显依赖性和层次性的检索方式。如图9.15所示，递归型检索的显著特点在于每一步都依赖于前一步的输出，并通过不断加深检索过程，逐步挖掘更深层次的信息。通常，递归型检索遵循类似树状的结构，每次检索都会基于一个重新改写的查询展开，从而精确地针对当前需要获取的知识进行检索。递归型检索还包含明确的退出机制，用以确保在满足终止条件时流程终止，避免无限递归。这种机制能够有效控制流程的深度和复杂性。在 RAG 系统中，递归型循环模式通过查询转换模块生成新的查询，以推动检索逐层深入。这种方式特别适合需要分步推理或分解复杂问题的任务场景，能够逐步定位相关信息并生成高质量的回答。

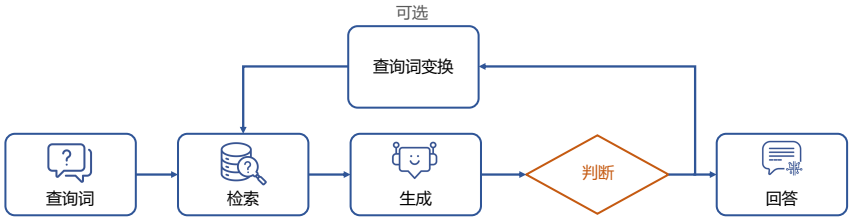

图 9.15　RAG flow 的递归型循环模式[421]

（3）自适应型（主动型）模式是一种超越传统被动检索模式的新兴模式，得益于大语言模型的强大能力。如图9.16所示，这种模式的核心思想类似于大语言模型智能体，通过动态调整检索流程，主动决定何时进行检索以及何时终止流程并生成最终结果。与传统的固定流程不同，自适应型检索具有更高的灵活性和智能性，能够根据任务需求实时调整策略。自适应型检索通常根据判断标准进一步细分为两种方法：基于提示的方法和基于指令微调的方法。前者通过设计动态提示对模型进行引导，而后者则利用指令微调的方法实现更精准的检索控制。这种模式特别适用于复杂任务或动态信息需求的场景，因为它能够智能判断流程的最佳执行路径，从而提高检索效率和生成质量。

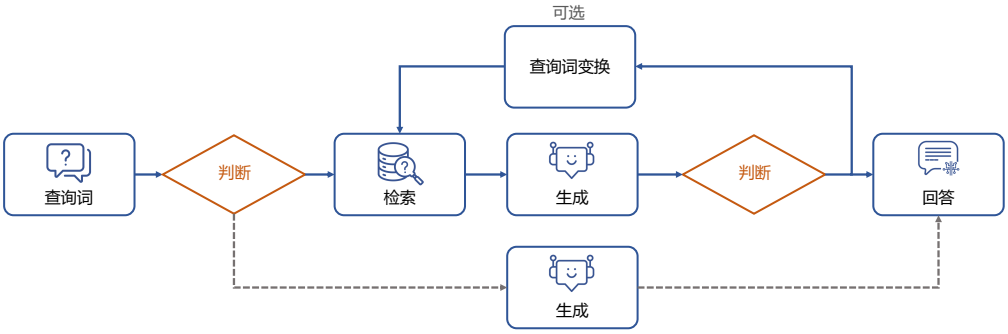

图 9.16 RAG flow 的自适应型循环模式[421]

9.4 RAG 系统训练与优化

通过对 Modular RAG 架构和 RAG 设计模式的分析可以发现，许多模块的功能都依赖于模型的能力，这些模块的效果也直接影响了系统的整体性能。例如，向量块的优化需要深入理解上下文的语义相关性，以保证文本块切分过程中保持语义相关度；查询转换模块需要将用户的自然语言查询转化为适合检索的查询表达式，确保检索系统能够找到最相关的文档；而在检索后的优化阶段，则需要使用重排序模型对返回的文档块进行重新排序，根据用户输入判断其相关性，以提供更准确的结果。

这些能力的实现通常依赖于模型的训练和优化。一方面，可以通过传统的小模型进行定制化训练，以针对特定任务和领域进行优化；另一方面，也可以直接利用大语言模型其强大的通用能力，尤其是在处理复杂语义理解、上下文关联性判断以及多轮交互等方面。此外，不同模块对模型能力的依赖程度也各不相同。例如，向量化模块需要借助预训练模型生成高质量的嵌入向量，以捕捉文本的深层语义特征；查询转换模块可能需要结合提示工程、或模型微调的方式，生成更精准的检索查询；而重排序模型则需要在结合用户输入和上下文的基础上，优化排序策略以提高最终输出的质量。因此，如何高效地选择、训练和集成这些模型，成为构建高性能 RAG 系统的关键。

本节将按照 Modular RAG 架构中模块划分，介绍典型的 RAG 系统中各模块所采用算法和优化方法。

9.4.1 文本嵌入模型微调

文本嵌入（Text Embedding）是一种将文本转换为固定维度向量（通常是高维浮点数组）的技术，旨在以数学形式捕捉语言的语义信息，并将其映射到向量空间中。通过深度学习模型（如 Word2Vec、GloVe、FastText，以及基于 Transformer 的模型如 BERT、Sentence-BERT 和 OpenAI 的 text-embedding-ada 等），文本的语义、语法及上下文特征能够被有效编码为向量表示。在 RAG

（Retrieval-Augmented Generation）系统中，文本嵌入表示是实现向量搜索的核心技术。

文本嵌入技术有很长的研究历史，大体上可以分为四个阶段：计数式嵌入（Count-based Embeddings）：这一阶段的方法包括词袋模型（Bag of Words, BoW）和 TF-IDF，用词频和逆文档频率来表示文本，但忽略了词语的语义和上下文信息，仅能反映基本的词汇相关性；静态词嵌入（Static Dense Word Embeddings）：代表性模型如 Word2Vec、GloVe 和 FastText，通过上下文生成固定的词向量。这一阶段捕捉了词语的语法和语义相似性，但每个词的向量是静态的，无法反映词义在不同上下文中的变化；上下文嵌入（Contextualized Embeddings）：这一阶段引入了上下文敏感的动态嵌入模型，如 GPT 和 BERT 等。这些模型通过双向或单向 Transformer 结构，生成能够根据上下文调整的词或句子向量，实现了对多义词和复杂语境的更深层次理解；通用文本嵌入（Universal Text Embeddings）：最新阶段致力于构建能适配多任务、多领域、多语言的统一模型。通过利用大规模多样化数据、合成数据生成以及大语言模型（LLMs）作为骨干网络，如 E5[434]、BGE[435]、Gecko[436] 等，通用文本嵌入模型可以在分类、检索、聚类等任务中表现出色，显著提升了跨任务和跨领域的泛化能力。

通用文本嵌入模型目标应对众多下游任务，文献 [437] 提出的 GTE 模型（General-purpose Text Embedding）引入了多阶段对比学习策略，并采用多样化的训练数据混合方式：在预训练阶段，使用未经任何筛选或清理的大量开源数据，通过无监督对比学习来学习基本的语言模式；在第二阶段，利用有监督微调，通过对比学习使用规模更小、质量更高的数据集对嵌入向量进行优化。

对于查询语句 q 所对应的一个相关（正例）文档 $d+$ 以及一组不相关（负例）文档 $D_- = \{d_1^-, d_2^-, \ldots, d_n^-\}$，InfoNCE 损失[53] 的定义如下所示：

$$L_{cl} = -\log \frac{\mathbf{e}^{\mathbf{s}(q,d^+)/\tau}}{\mathbf{e}^{\mathbf{s}(q,d^+)/\tau} + \sum_{i=1}^{n} \mathbf{e}^{\mathbf{s}(q,d^-)/\tau}} \tag{9.10}$$

其中，$\mathbf{s}(q,d)$ 通过文本 q 和 d 的嵌入向量（$q = E(q)$ 以及 $d = E(d)$）之间的向量距离来估计这两段文本之间的相似度。

GTE 模型中，给定一批正例文本对样本 $\{(q_1,d_1),(q_2,d_2),\ldots,(q_n,d_n)\}$，作者提出一种改进的对比损失，如下所示：

$$L_{icl} = -\frac{1}{n}\sum_{i=1}^{n} \log \frac{\mathbf{e}^{\mathbf{s}(q_i,d_i)/\tau}}{Z} \tag{9.11}$$

$$Z = \sum_{j} \mathbf{e}^{\mathbf{s}(q_i,d_j)/\tau} + \sum_{j\neq i} \mathbf{e}^{\mathbf{s}(q_i,q_j)/\tau} + \sum_{j} \mathbf{e}^{\mathbf{s}(q_j,d_i)/\tau} + \sum_{j\neq i} \mathbf{e}^{\mathbf{s}(d_j,d_i)/\tau} \tag{9.12}$$

其中，Z 采用余弦相似度作为相似度度量 $s(q,d)$。GTE 使用 BERT 等预训练语言模型进行初始化，通过对语言模型生成的上下文词元表示进行平均池化来获取文本嵌入向量。

GTE 模型在预训练阶段，使用了约 8 亿对无标注的文本对，数据来源多样，包括网页数据（如

Common Crawl 和 MS MARCO 文档，标题作为查询，正文作为文档）、学术论文（如 PubMed 和 arXiv，标题与摘要配对）、超链接（如 Wikipedia 和引用文本配对）、社交媒体（如 Reddit 的帖子与评论对）、知识库（如 WikiPedia 和 DBPedia 的实体和描述对）、社区问答网站（如 StackExchange 和 WikiHow 的标题与正文、问答对）、新闻、代码数据以及其他来源（如商品评论和 Google 搜索日志）。在微调阶段，数据进一步聚焦于特定任务，包括网页搜索（如 MS MARCO 检索任务中的正负样本对）、开放式问答（如 Natural Questions 和 Trivia QA，通过检索系统生成困难负样本）、自然语言推理（如 MNLI 和 SNLI 的推断与矛盾对）、事实验证（FEVER 训练集）、语义复述（如 Quora 和 StackExchange 的复述任务）以及多个领域和任务的其他数据集（如 MEDI 和 BERRI）。这种多样化且精心设计的数据分布为模型提供了广泛的语义理解能力，同时通过微调使其能够在特定任务中表现出色。

虽然通用文本嵌入已经有非常的好效果，但是针对特定领域的微调对于提升检索质量依然有非常重要的影响。通过微调，模型能够更准确地理解查询的语境和细微差异，从而提高检索阶段的效果。具体而言，微调能够增强模型的语义匹配能力，使其生成更具语境感知的嵌入，这不仅能更有效地匹配查询与潜在文档，还能显著提升检索内容的相关性。对于特定领域的数据进行微调，可以使模型更好地掌握领域专有的术语、风格和知识，生成更加精准和专业的内容。特别是在处理稀有查询时，微调可以充分利用领域知识，有效应对罕见或特殊表述的查询，这对于医疗、法律和教育等专业领域尤为重要。

文献 [438] 提出了专门针对医学文档检索的框架 REMED，其中 EM-FT 模型通过高效的嵌入式微调方法，对预训练模型中的医学句子表示进行端到端微调，从而提高医学检索性能。作者选用 m3e-base[439] 和 e5-base-v2[434] 作为嵌入模型的基线。EM-FT 方法结合了对比学习作为损失函数，以优化模型性能并准确捕捉查询和相关文档之间的相似性，使得与查询相关的文档比不相关的文档更接近，如下公式所示：

$$L(W) = L(q, p_1^+, p_2^+, \cdots, p_n^+, p_1^-, p_2^-, \cdots, p_m^-) \tag{9.13}$$

$$L(W) = -\log \frac{\sum_{i=1}^{n} \mathbf{e}^{\left(\text{sim}(q, p_i^+)\right)}}{\sum_{i=1}^{n} \mathbf{e}^{\left(\text{sim}(q, p_i^+)\right)} + \sum_{j=1}^{m} \mathbf{e}^{\left(\text{sim}(q, p_j^-)\right)}} \tag{9.14}$$

其中 $L(W)$ 表示通过训练模型参数 W，最大化正样本段落相对于查询 q 的相关概率，并最小化负样本段落的相关概率，q 表示输入查询。p_i^+ 是与查询相关的正样本段落，p_j^- 是与问题不相关的负样本段落。同样也是采用余弦相似度 $sim(q,p) = \cos(E(q), E(p))$ 作为评分函数来衡量查询 q 和段落 p 之间的匹配程度。EM-FT 模型由两个核心组件组成：嵌入骨干网络（Embedding Backbone）和可训练 EM 头（Trainable EM Head）。嵌入骨干网络负责处理输入的文本数据，而可训练 EM 头则通过归一化层、两个线性层和激活函数来实现高效的文本相似度检索。

为了训练更能适应医疗领域的文本嵌入表示，文献 [438] 构建了 Medical Menu Dataset (MMD)

和 Medical Paper Dataset (MPD)。MMD 是一个综合且可靠的医学信息检索评估基准，专注于医疗领域的检索系统性能测试。该数据集的数据来源于权威的“WHO Medicine”数据库以及“国家药典”中所有药物信息，包含超过 20 万条记录。MPD 是一个从美国国家生物技术信息中心（NCBI）采样 1,000 篇医学论文构建而成的数据集。为确保分析的准确性和可靠性，MPD 经过了一系列预处理和清洗操作，排除了不符合研究标准的文献（如非正式会议演讲和非同行评审的报告），并移除了表格数据和不规范的数学公式。清洗后的文档被分割为固定长度的文本段（最大序列长度为 768），以适应嵌入模型的输入要求，同时保留足够的上下文信息。最终 MPD 包含 886 篇论文，共 79,966 条数据。实验结果证明，EM-FT 方法在 MMD 上的召回率和精度分别提高了 3.2%-6.0%，在 MPD 上的召回率和精度分别提高了 14.4%-42.6%。在一定程度上也说明，针对特定领域对文本嵌入模型进行微调很有必要。

9.4.2 查询优化

如前所述，RAG 系统在处理用户查询时，需要对查询优化进行深入改进，以应对多种复杂挑战。对于简单查询，例如日常问候等无需上下文支持的情况，模型应避免执行不必要的信息检索，直接生成答案，从而减少无关上下文对响应质量的影响。对于复杂查询，直接使用原始查询进行检索通常难以获取足够的相关信息。模型需要首先将复杂查询拆解为可解答的子查询，分别检索与其相关的信息，并整合子查询的结果，生成对原始查询的完整回答。而对于多义性较强的模糊查询，直接检索原始查询往往无法提供全面的答案。模型需通过识别用户意图来澄清查询内容，并构建精准的检索请求，获取相关信息后生成细致且全面的响应。通过优化查询流程，RAG 系统不仅能够提升检索效率，还能显著增强模型在复杂场景中的适应能力和表现。

针对上述问题，文献 [440] 提出了 RQ-RAG 算法，旨在通过动态优化查询以提升检索增强生成的效果。该方法基于 7B 规模的 Llama2 模型，采用端到端训练，使其能够通过重写、分解和消除歧义来动态优化搜索查询。为了训练模型具备上述功能，核心是构建与推理过程相匹配的训练数据。为了生成高质量的大规模数据，文献 [440] 采用了与 Self-RAG[431] 和 SAIL[441] 类似的方法，设计了一套自动化的数据生成流程，以优化查询、检索信息并生成精确的响应，同时减少人工干预所需的资源和时间成本。

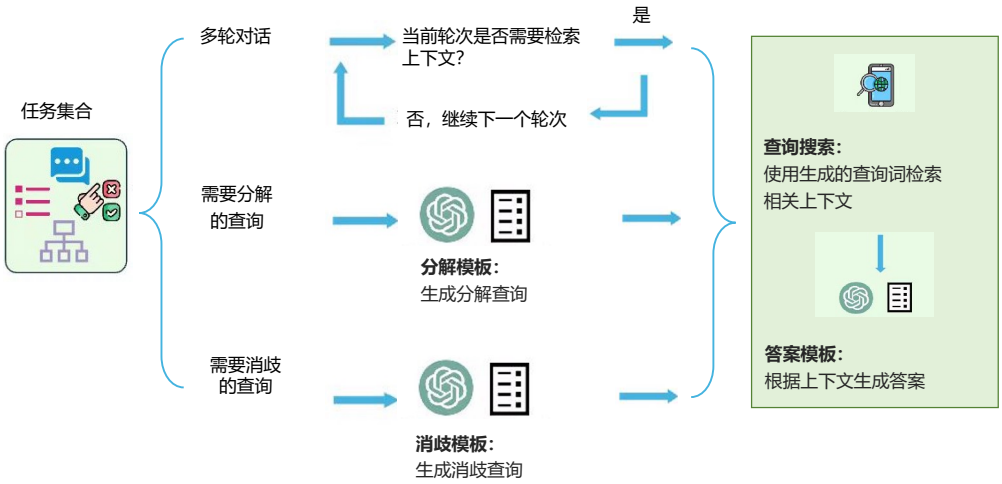

图 9.17 RQ-RAG 数据构造的流程[440]

数据构造整体流程如图9.17所示，整个流程分为以下几个关键步骤：

（1）从任务池中收集代表性任务，并将其分类为三种类型（如消歧查询、复杂查询分解等），根据任务特性，每个数据集对应特定的数据类型。这一步通过任务的特性和需求进行分类，确保数据生成流程的针对性。

（2）对于每种任务类型，使用预定义的提示模板调用 ChatGPT 生成优化后的查询。提示模板根据任务类型的不同进行了定制，例如针对模糊查询的提示会强调消除歧义，而针对复杂查询的提示会引导模型进行分解。生成的优化查询被用于从外部数据源检索相关信息，检索过程以 DuckDuckGo 为主要搜索引擎，其他搜索工具（如 Bing）作为补充。

（3）使用 ChatGPT，根据优化后的查询及其对应的检索上下文生成响应。在这一阶段，ChatGPT 被提示根据上下文信息生成与查询高度相关的回答，同时避免冗余和噪声信息对响应质量的干扰。整个流程通过不断重复，最终生成了约 40,000 条数据实例。

RQ-RAG 所使用的任务池涵盖了多种代表性任务，确保模型能够适应不同场景需求。这些任务包括单跳问答任务（如 Arc-Easy/Arc-Challenge[442] 和 OpenbookQA[443]），用于测试模型的基础推理能力；多跳问答任务（如 HotpotQA[444] 和 Musique[445]），要求模型整合多步信息以推导答案；以及歧义问答任务（如 ASQA），评估模型处理多义性问题的能力。此外，为了提升模型的通用能力，还引入了指令跟随任务，包括 LIMA[446]、WizardLM[447]、Open-Orca[448]、OpenAssistant[200] 和 GPT4-Alpaca[35]，这些任务通过多样化的场景训练模型理解和执行自然语言指令的能力。最终，任务池共收集了 42810 个实例，为模型的训练提供了丰富且全面的支持。

在对训练语料库进行标注之后，采用标准的自回归方式来训练大语言模型，其目标如下公式

所示：

$$L = \max_{M} E_{(x,y)\sim D}[\log p_M(y|q_1, d_1, d_2, \ldots, q_i, d_i, x) \tag{9.15}$$

其中，L 代表试图最大化的概率值，M 表示模型参数，期望 $E_{(x,y)\sim D}$ 是对数据集 D 求平均，$p_M(y|q_1, d_1, q_2, d_2, .$ 表示在给定输入 x、第 i 步经过优化的查询 q_i 以及检索到的文档 d_i 的情况下，模型 M 生成回复 y 的概率。

RQ-RAG 在推理过程中采用了一种树形解码策略，其具体流程如图9.18所示。在每个时间步，模型可以根据需要对查询进行重写、分解、消除歧义，或直接生成回答。通过特殊标记的引导，该策略能够控制解码路径的扩展，并以“生成 → 检索 → 生成 → 检索 →……→ 答案”的循环过程逐步展开。在每次迭代中，模型会根据任务需求生成不同类型的搜索查询，例如重写、分解或消歧查询。这些查询将被用于检索与其对应的上下文信息，从而形成不同的解码路径。基于设定的探索宽度和深度范围，RQ-RAG 能够生成多条候选轨迹，通过逐步迭代的方式全面探索潜在答案的空间，为最终的响应提供更丰富的支持。

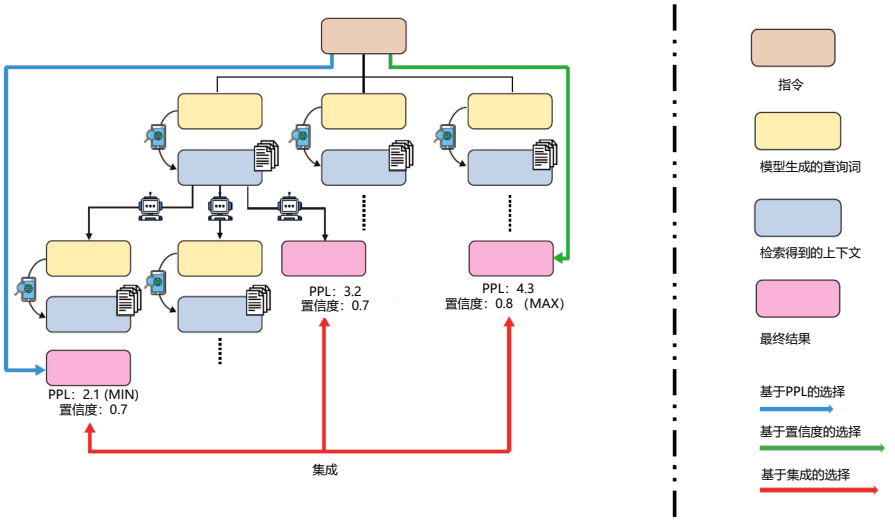

图 9.18　RQ-RAG 解码策略流程[440]

如何从这些轨迹中选取最合适的路径是 RQ-RAG 系统中的关键问题之一。令 p_M 表示一个参数为 M 的大语言模型，$[R_1, R_2, \ldots, R_n]$ 表示 n 条轨迹，其中每条轨迹都包含一个序列，记为 $[X, Y]$。其中，X 是输入提示，Y 是由 $Z_1, Z_2, \ldots, Z_i$（每个 Z_i 都是查询和检索到的上下文的组合）组成的 i 个中间步骤，以及最终答案 Y_{final} 的拼接结果。针对这一问题，RQ-RAG 方法提出了三

种不同的采样策略，具体如下：

（1）基于困惑度（PPL）的选择：从生成的所有轨迹中选择困惑度（PPL）最低的轨迹 R_{final}，其定义为：$R_{\text{final}} = \arg\min_{R_j \in \{R_1,\ldots,R_n\}} \text{PPL}(R_j)$，其中 $\text{PPL}(R) = \exp\left(-\frac{1}{L}\sum_{t=1}^{L} \log p_M(Y_t|X, Y_{<t})\right)$，这里 L 是模型输出的总长度，$p_M(Y_t|X, Y_{<t})$，是语言模型在输入为 X，且以先前已经生成的输出 $Y_{<t}$ 作为条件时，生成第 t 个标记 Y_t 的概率情况。

（2）基于置信度的选择：选择对最终答案 Y_{final} 具有最高置信度的轨迹 R_{final}（这与基于困惑度的选择有所不同，后者评估的是全部生成的输出），即 $R_{\text{final}} = \arg\max_{R_j \in \{R_1,\ldots,R_n\}} \text{Conf}(R_j)$，其中 $\text{Conf}(R) = \sum_{t=l} \log p_M(Y_t|X, Z_1, \ldots, Z_i, Y_{<t})$，这里 t 从 l 开始，l 是最终答案 Y_{final} 的起始位置。

（3）基于集成的选择：选择累积置信度得分最高的结果作为最终输出，可以表示为：$Y_{\text{final}} = \arg\max_y \sum_{i:Y_i=y} \text{Conf}(Y_i)$。其中，最终结果 Y_{final} 是所有候选结果中置信度分数累积最大的一项，通过对所有候选结果 Y_i，取值等于 y 的置信度分数 $\text{Conf}(Y_i)$ 进行累加求和，确定最佳答案。

9.4.3 幻觉感知的生成模型优化

大模型幻觉指的是大语言模型生成的内容中出现与事实不符、缺乏依据或与输入信息相矛盾的表述。在实际应用中，即使采用检索增强生成（RAG）方法，大语言模型仍然可能出现幻觉问题，例如对检索到的内容进行错误或扭曲的解释，这在高信任场景中带来了显著风险。

文献 [449] 提出了一种专门针对检索增强生成中幻觉问题的方法，Hallucination Aware Tuning（简称 RAG-HAT）。该方法通过训练幻觉检测模型，识别出幻觉并给出易于理解的解释，说明幻觉产生的位置和原因，以及提供防御性建议。利用这些检测结果，特别是幻觉描述，借助 GPT-4 Turbo 对包含幻觉的 RAG 输出进行重写，以去除幻觉内容。随后，原始输出和修正后的输出被用于构建偏好数据集，通过直接偏好优化（Direct Preference Optimization，DPO）方法对大语言模型进行训练，从而有效降低模型生成幻觉的概率，同时提升回答质量。

RAG-HAT 在构造幻觉检测方法时，采用了基于选择性采样的训练数据构建策略。在 RAGTruth[450] 数据集的基础上，虽然该数据集标注了幻觉文本的具体片段，但缺乏对幻觉的详细描述，因此 RAG-HAT 借助 GPT-4 Turbo 自动生成幻觉描述，以支持检测模型的训练。这些描述包括三部分内容：幻觉的二元标签（标识句子是否包含幻觉）、幻觉发生的位置和原因的详细解释，以及防御性建议（Defensive Advice）。防御性建议明确指出文本中可能导致幻觉的模糊表述，并提供改进建议，从而帮助减少分类边界的不确定性，降低幻觉的发生率。此外，RAG-HAT 借鉴了自举式训练（Bootstrapping-style Training）和拒绝采样的策略，对 GPT-4 的输出进行多轮评估与再生成，以确保生成数据的质量与准确性。

在检测模型的训练过程中，RAG-HAT 采用了两阶段策略。第一阶段专注于训练模型输出幻觉的预测标签，完成基础的幻觉检测任务；第二阶段通过使用 LoRA 微调，使模型能够基于预测标签生成幻觉的详细解释，包括幻觉描述以及防御性建议。在推理时，两阶段模型以级联方式应用，先检测幻觉，再生成解释性描述。这种训练策略不仅显著提升了幻觉检测的精度，还增强了模型

在处理边界案例时的解释能力和鲁棒性。

RAG-HAT 采用 DPO 方法进行模型训练，通过构建成对的偏好数据集，指导大语言模型生成更少幻觉内容的回答。在回答重写阶段，针对包含幻觉的原始回答，结合生成的幻觉解释内容，利用 GPT-4 Turbo 对其进行重写，去除幻觉并生成“优选”（Chosen）样本。而对于被判定为优质的回答，则通过防御性建议限定重写范围，仅针对特定句子进行优化，以避免引入新的幻觉内容。此外，重写后的回答通过幻觉检测模型进行验证，确保其准确性，如发现仍存在幻觉，则重复重写过程，直至生成高质量的样本，保证数据集的完整性和可靠性。

为进一步提升模型的回答质量，RAG-HAT 还在偏好数据集中引入了“过于谨慎惩罚”（Overly Cautious Penalization, OCP）策略。由于模型在训练后可能倾向于通过缩短回答来降低幻觉率，从而影响回答的内容丰富性，OCP 随机从“优选”样本中删除一个句子以生成“拒绝”（Rejected）样本，鼓励模型在减少幻觉的同时保持回答的内容完整性。此外，为扩展训练数据规模，RAG-HAT 通过自动化流程将 XSum[451] 数据集和 Marco[452] 数据集中的样本转换为新的回答，并与 RAGTruth 数据集中的答案共同组成偏好对，确保“拒绝”样本能够准确反映模型的输出分布。最终，该方法共生成了 19,721 对“优选/拒绝”样本，用于 DPO 训练，从而有效平衡了减少幻觉与回答质量之间的需求，提高了模型的实际应用表现。

9.4.4 重排模型优化

RAG 系统中通过检索模块从知识库中获取与输入问题相关的信息。然而，初步检索的结果通常基于简单的相关性度量（如 BM25 或密集向量检索），这些方法需要综合考虑效果和效率，所采用的方法无法完全捕捉输入问题的语义意图，从而导致噪声或不完全相关的文档被返回。重排模型的引入旨在针对检索到的候选文档进行精细排序，优先选择那些与输入问题更相关的文档，为生成模型提供更高质量的上下文。

得益于大语言模型在语言理解、生成、交互和推理等方面的卓越表现，利用大语言模型进行文档重排序受到了很多关注。这些方法通常将大语言模型用作点估计器[417] 或列表重排序器[453, 454]。尽管这些方法能够灵活定义文档相关性，并支持零样本场景下的操作，但它们在决策过程中缺乏中间分析步骤。在需要复杂推理的场景中，这种局限性会影响模型的性能和可解释性。此外，列表重排序器还面临显著的计算挑战，主要源于上下文长度的限制。当需要同时处理多个文档时，列表重排序器往往不得不牺牲单个文档的长度，以满足整体处理需求。这种权衡进一步限制了其在高复杂度任务中的表现。

为了解决现有方法在复杂推理场景中的局限性，JudgeRank[455] 提出了一种的零样本点式重排序方法，专为需要深入推理的文本检索任务设计。JudgeRank 利用高度通用的提示引导经过指令微调的大语言模型，通过显式的推理步骤来得出最终的相关性判断。这种方法通过逐步推理的方式增强了大模型在推理密集型任务中的表现。

JudgeRank 的工作流程包括三个关键步骤：1）问题分析：模型通过提示识别查询中的核心问

题，从而专注于关键问题并过滤掉无关的上下文；2）文档摘要：对每个候选文档生成抽取式摘要，并解释文档如何回应查询；3）相关性判断：基于之前的分析，模型对文档的相关性进行最终判断。这一过程模拟了人类回答问题的思维方式：先快速浏览文档，找到与问题相关的部分，再仔细阅读这些内容以得出答案。

JudgeRank 在问题分析部分所使用的提示词如下所示：

```
You will be presented with a/an query name.
Your task consists of the following step:

1. Analyze the {query name}:
- Carefully read each sentence of the {query name}.
- Identify the core problem or question being asked.

Here is the {query name}:
{query}
```

JudgeRank 在文档摘要部分所使用的提示词如下所示：