```
You will be presented with a/an {query name}, an analysis of the query, and a/an {doc name}.

Your task consists of the following steps:
1. Analyze the {doc name}:
- Thoroughly examine each sentence of the {doc name}.
- List all sentences from the {doc name} that {definition of relevance} the {query name}.
- Briefly explain how each sentence listed {definition of relevance} the {query name}.

2. Assess overall relevance:
- If the {doc name}, particularly the relevant sentences (if applicable), {definition
  of relevance}
the {query name}, briefly explain why.
- Otherwise, briefly explain why not.

Here is the {query name}:
{query}

Here is the analysis of the {query name}:
{query analysis}

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

JudgeRank 在相关性判断部分所使用的提示词如下所示：

```
You will be presented with a/an {query name}, an analysis of the {queryname}, a/an {doc name},
and an analysis of the {doc name}.

Your task is to assess if the {doc name} {definition of relevance} the {query name} in one word:
- Yes: If the {doc name} {definition of relevance} the {query name}.
- No: Otherwise.

Important: Respond using only one of the following two words without quotation marks: Yes or No.

Here is the {query name}:
{query}

Here is the analysis of the {query name}:
{query analysis}

Here is the {doc name}:
{doc}

Here is the analysis of the {doc name}:
{doc analysis}
```

在对文档相关性进行判断后, 文档评分的合成方法旨在通过多种策略对文档进行重新排序, 以提高检索结果的相关性。这些方法包括离散版本、连续版本和混合版本。在离散版本中，文档根据模型的判断被划分为“相关”（输出为“是”）和“不相关”（输出为“否”）两类。对于每一类文档，保留初始检索排名的相对顺序，即相关文档始终排在不相关文档之前。虽然这种方法简单直观，但其性能高度依赖于提示的设计和第一阶段检索的质量。

为了克服离散方法的局限性，连续版本利用模型输出的“是”概率（p_y）和“否”概率（p_n）对文档进行更细粒度的评分。具体来说，评分函数通过归一化 p_y 和 p_n 的值来计算文档的相关性得分：$S(d) = \frac{p_y}{p_y+p_n}$，从而确保不同文档的评分具有可比性。根据这些得分，所有文档被重新排序，得分越高的文档排名越靠前。与离散版本相比，连续版本能够更精确地捕捉文档的相关性梯度，适用于需要更细腻排序的场景。

混合版本进一步结合了连续版本的概率评分和第一阶段检索中的 BM25 分数，通过加权求和的方式生成综合评分。具体地，最终评分由概率得分 S_{prob} 和 BM25 分数 S_{BM25} 按照权重系数 α 进行加权：$S = \alpha S_{\text{prob}} + S_{\text{BM25}}$，综合了推理能力和表层匹配的优点。混合版本通过模型集成的方式，兼顾深层语义推理和表层匹配效果，在实际应用中表现出更强的稳定性和适用性。

9.4.5 检索与生成联合优化

文献 [456] 提出了 RankRAG 方法，利用单个大语言模型完成重排序和答案生成。RankRAG 通过两阶段微调策略：通用指令微调以及排序与生成指令调优。不仅优化了语言模型的生成能力，还赋予其上下文排序能力。RankRAG 方法的训练和推理流程如图9.19所示。

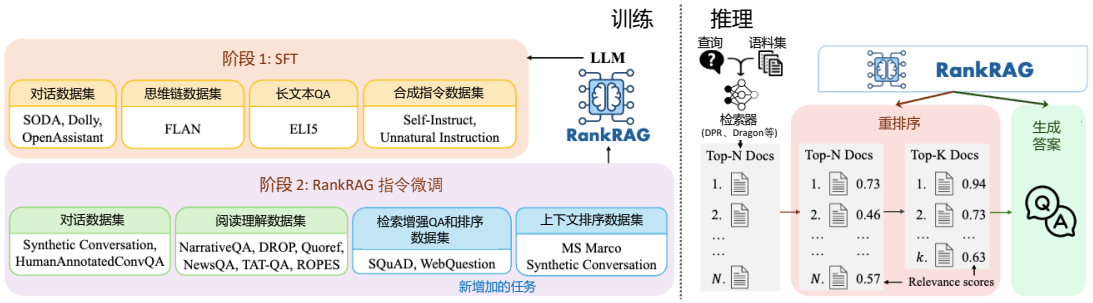

图 9.19 RankRAG 方法流程[456]

在第一阶段，RankRAG 通过有监督微调提升语言模型的基本指令遵循能力。使用的数据包括高质量的指令遵循数据集，例如 OpenAssistant、Dolly、SODA 以及长文本问答数据集 ELI5 等，总计 128,000 样本。微调过程中，模型采用多轮对话格式，将用户与助手的历史对话作为上下文，仅对助手的最后一个响应计算损失。这一阶段为模型奠定了基础，使其能够更好地理解和执行指令。

第二阶段的指令调优专注于增强模型的检索排序和生成能力。本阶段训练结合了五种数据类型：（1）第一阶段的通用有监督微调数据，用于保持模型的指令遵循能力；（2）上下文丰富的问答数据，用于训练模型从复杂上下文中生成答案；（3）检索增强的问答数据，通过结合标准上下文和 BM25 检索到的上下文，训练模型在生成答案时处理混合上下文的能力；（4）上下文排序数据，利用 MS MARCO 排序数据和合成会话数据，训练模型判断单个上下文的相关性；（5）检索增强的排序数据，通过多上下文任务训练模型同时判断多个上下文的相关性。这种任务设计使模型能够更稳健地处理检索结果中的噪声，提升了对上下文的筛选能力。

RankRAG 的核心创新在于将各种任务标准化为统一的 QA 格式，即 (x, c, y)，其中 x 表示问题，c 是上下文，y 是目标输出。例如，对于检索增强的排序任务，问题可以表述为“针对问题 < 问题 >，从上下文中找到所有相关段落。”这种标准化方法不仅简化了多任务学习，还通过知识迁移互相增强不同任务。这种方法只增加少量排序数据，即赋予模型排序能力，同时提高了生成任务的表现。

RankRAG 包含一个重排序步骤，其推理流程遵循“检索-重排序-生成”的模式，具体包括以下三个阶段：（1）检索阶段：检索器 R 首先从语料库中检索出与问题相关的前 N 个上下文，为后续步骤提供候选信息。（2）重排序阶段：利用 RankRAG 模型计算问题与检索到的 N 个上下文之

间的相关性得分。相关性得分被视为生成正确答案（True）的概率。根据相关性得分对上下文进行重新排序，仅保留前 k 个上下文（$k \ll N$），这些上下文被视为最相关的信息源。(3）生成阶段：将保留的前 k 个上下文与问题连接后输入 RankRAG 模型，用于生成最终答案。

9.5 RAG 系统评估

检索增强生成系统通过将信息检索与生成模型相结合，在知识密集型任务中展现出了显著的应用潜力。然而，正是由于其复杂的混合结构与对动态知识的依赖，使得对其性能进行全面评估面临诸多挑战。为了科学、系统地评估 RAG 系统的能力，不仅需要分别考察检索组件与生成组件的独立表现，还需关注两者之间的协同作用。同时，为确保评估的准确性与全面性，应结合多维度的评估指标与多样化的数据集设计合理的评估方案。

本节将围绕 RAG 系统的评估展开，详细探讨其评估过程中所面临的挑战、评估数据集的选取与设计以及评估指标的制定。

9.5.1 RAG 评估的挑战

RAG 系统通过结合外部知识库的检索与生成模型的生成能力，有效解决了传统生成模型中内容缺乏事实依据的问题。然而，其复杂的“检索-生成”架构使得评估变得尤为重要，不仅需要考察检索和生成组件的独立性能，还需关注两者的协同作用，以及系统在动态知识更新中的适应性与泛化能力。此外，RAG 系统在不同任务场景中的表现差异、多样化的应用需求以及对用户体验的影响（如响应速度、生成内容的准确性和可读性）都凸显了构建全面评估框架的必要性，以及 RAG 评估所面临的挑战。

检索模块是 RAG 系统的核心部分，负责从庞大的外部知识库中提取与用户查询相关的信息。检索模块的评估面临多重挑战。首先，知识库的广度与动态性使得评估复杂化。RAG 系统通常依赖多样化的知识来源，包括结构化数据库、维基百科页面甚至整个互联网，这些知识库的内容会随着时间和领域的变化而更新。因此，需要设计评估指标，以衡量系统在不同时间点和知识领域中检索有效性的稳定性和准确性。其次，检索内容的质量直接关系到生成组件的表现。除了评估相关性外，还需考察检索结果的准确性和可靠性。低质量或误导性的检索内容可能对生成结果产生负面影响，因此对检索内容的筛选和质量控制至关重要。此外，现有的检索指标（如精确率和召回率）无法全面反映 RAG 系统的特性。这些传统指标缺乏对检索结果是否能够满足后续生成需求的评估能力，也无法量化检索与生成之间的协作效果。

生成模块通过大语言模型对检索结果进行加工，以生成连贯且与查询相关的回答。生成模块的评估同样面临诸多挑战。首先，生成内容必须具有真实性与一致性，这意味着生成的回答需要忠实于检索到的信息，同时满足用户的查询需求。这种真实性与一致性的评估需要结合检索结果，而不仅仅依赖生成内容本身。其次，开放式任务的主观性增加了评估难度。在开放域问答或创造性文本生成任务中，可能不存在唯一正确的答案，不同评测者对高质量生成的定义也存在差异，这

使得评估结果容易受到主观因素的影响。此外，生成内容的质量评估需要覆盖多个维度，包括准确性、连贯性、流畅性和可读性等。这些维度的多样性要求设计更加细致和全面的评测指标，以全面反映生成组件的表现。

RAG 系统的整体性能评估需要超越对检索和生成组件的单独考察，重点关注两者之间的协同作用。检索结果的质量会直接影响生成结果，而生成组件的表现也取决于其对检索内容的有效利用。因此，评估需要量化检索对生成的实际贡献，并分析两者在不同任务中的交互效果。此外，在实际应用中，RAG 系统的响应能力同样重要。例如，用户通常关注系统处理模糊查询的能力、响应速度以及在多轮对话中的表现。这些实际应用场景中的关键因素往往被传统评测框架所忽略，因此需要在整体评估中引入新的指标和方法，以全面衡量 RAG 系统在真实场景中的实际效用和用户体验。

9.5.2 评估目标

评估目标是 RAG 系统性能评估的核心，直接决定了评估框架的设计方向与具体实施方式。评估目标需要清晰地定义检索组件、生成组件及整体系统的性能衡量标准，同时能够全面覆盖系统在不同任务场景中的表现。根据 RAG 系统的“检索-生成”结构，评估目标可分为针对检索的评估、针对生成的评估，以及面向整体系统的协同能力评估。

1. 检索模块的评估目标

检索模块是 RAG 系统的基础，其主要任务是从知识库中提取与用户查询相关的信息，为生成模块提供支持。在评估检索组件时，需要明确以下几个关键目标：

（1）相关性（Relevance）：检索组件的首要目标是确保其返回的文档与用户查询高度相关。相关性评估旨在衡量检索出的文档是否能够准确反映用户查询所需要的信息。例如，在问答任务中，检索出的文档是否包含回答问题所需的事实或背景知识。相关性通常通过计算检索结果与查询之间的匹配程度来评估，可以使用指标如精确率（Precision）和召回率（Recall）。

（2）准确性（Accuracy）：除了相关性，检索结果的准确性也至关重要。准确性评估需要考察检索到的文档是否在信息上是可靠的，是否包含错误、误导性内容或低质量信息。检索组件返回的错误信息可能直接导致生成组件生成不真实的回答，因此检索结果的准确性对整体系统的性能至关重要。

（3）覆盖率与多样性（Coverage and Diversity）：在某些任务中，用户的查询可能涉及多方面的信息需求。因此，检索组件需要确保其检索结果能够全面覆盖查询的不同维度，同时避免信息冗余。多样性评估旨在衡量检索结果是否包含多样化的视角或信息来源，尤其是在处理开放域问答或多轮对话时，这一点尤为重要。

（4）动态适应性（Dynamic Adaptability）：由于 RAG 系统依赖动态更新的知识库（如互联网爬取的数据），检索组件需要能够快速适应知识库的变化。动态适应性评估目标在于衡量检索组件是否能够在知识库更新后，及时检索到最新的相关信息。例如，在实时新闻问答场景中，检索结

果能否反映最新的事实将直接影响系统的有效性。

（5）排序能力（Ranking Ability）：检索组件通常返回一组潜在相关的文档，并根据相关性进行排序。评估其排序能力的目标是衡量系统是否能够将最相关的文档排在前面，这对生成组件的效率和性能有直接影响。排序能力通常通过排名指标（如平均倒数排名 MRR 和平均精确率 MAP）进行测量。

2. 生成模块的评估目标

生成组件的任务是利用检索结果，根据用户的查询生成连贯、准确且相关的回答。在评估生成组件时，需要明确以下几个关键目标：

（1）相关性（Relevance）：生成内容需要与用户查询保持高度相关。这不仅要求生成的回答能够回答用户的问题，还要求回答内容的范围与用户需求一致。例如，在开放式问答场景中，评估生成组件是否能够生成与查询语义一致的内容是关键目标之一。相关性通常通过人工评估或自动化指标（如 BLEU、ROUGE 等）来衡量。

（2）真实性与忠实度（Faithfulness）：RAG 系统的一个重要优势在于减少生成“幻觉”（hallucination）。因此，生成组件需要确保其输出内容忠实于检索到的信息，即生成的回答必须基于检索到的事实，而不是凭空捏造。真实性评估目标在于衡量生成内容是否准确反映了检索结果中的信息，避免出现事实错误或误导性内容。

（3）正确性（Correctness）：在许多任务中，生成的回答需要与给定的参考答案（Ground Truth）保持一致。正确性评估目标旨在衡量生成内容与标准答案之间的一致性，特别是在有明确答案的任务（如问答或填空任务）中。这通常通过自动化指标（如 F1 分数或精确匹配率）来衡量。

（4）连贯性与流畅性（Coherence and Fluency）：生成内容的连贯性和流畅性是评估生成组件的重要目标之一。连贯性指回答内容是否逻辑通顺，是否能够完整表达查询的意图；流畅性则关注语言表达是否符合自然语言的语法和用法。这些目标通常通过人工评估或语言模型的评分机制来实现。

（5）生成内容的多维度要求（Multi-Dimensional Requirements）：生成组件的评估需要覆盖多个维度，包括内容的可读性、丰富性和结构化程度。例如，在生成复杂文档摘要或表格形式的结构化内容时，需要评估生成结果是否符合预定义的格式要求。这些多维度的评估目标能够更全面地反映生成组件的表现。

（6）开放性任务的适应能力（Adaptability to Open Tasks）：在开放式生成任务中（如创造性写作或长文本生成），不存在唯一标准答案。评估目标需要更加灵活，能够衡量生成内容在语义层面的多样性与创新性，同时确保其与查询的核心意图保持一致。

3. 整体系统的评估目标

RAG 系统的整体性能不仅取决于检索和生成组件的独立表现，还需要关注两者之间的协同作用。整体系统的评估目标包括：

（1）协作效果（Collaboration Effectiveness）：整体系统的核心目标在于检索与生成组件的协作能力。评估需要量化检索结果对生成内容质量的贡献，以及生成组件如何利用检索内容来提升回答的准确性和相关性。

（2）任务完成度（Task Completion Rate）：在实际应用中，RAG 系统的整体目标是完成特定的任务，如回答用户问题或生成摘要。任务完成度评估目标在于衡量系统是否能够在特定任务中生成符合用户需求的高质量输出。

（3）用户体验（User Experience）：整体系统评估还需要包括对实际应用场景的考量，例如系统的响应速度、对模糊查询的处理能力、多轮对话中的表现，以及输出内容的可读性和实用性。这些目标直接关系到 RAG 系统的用户体验，是衡量系统整体表现的重要维度。

（4）鲁棒性与容错能力（Robustness and Fault Tolerance）：RAG 系统需要在面对噪声、不完整或不明确的查询时仍能生成有意义的回答。鲁棒性评估目标在于衡量系统在处理复杂或异常输入时的表现，以及系统在信息不足或不确定的情况下是否能够拒绝生成错误回答。

9.5.3 评估数据集

评估数据集是 RAG 系统性能评估中的关键组成部分，其质量和多样性直接影响评估结果的准确性和全面性。在评估 RAG 系统时，数据集的选择与构建需要兼顾系统的检索能力、生成能力以及整体协作表现。现有评估数据集的来源和构造方法多种多样，既包括基于已有资源的数据集，也包括为特定评估目标生成的全新数据集。不同的基准系统选择了不同的数据集策略，以适应各自的评估需求和应用场景。

1. 基于现有资源的数据集

许多评估框架依赖于已有的成熟数据集，如 KILT（Knowledge Intensive Language Tasks）基准[457] 和 SuperGLUE[458] 数据集。这些数据集涵盖了多种知识密集型任务。例如：Natural Questions (NQ)[459] 提供开放域问答任务的数据，测试系统对自然语言查询的回答能力。HotpotQA[444] 包含多跳问答任务，要求 RAG 系统能够从多个文档中综合信息来回答复杂问题。FEVER[460] 专注于事实验证任务，评估系统对检索信息支持或反驳查询的能力。MultiRC[?] 和 ReCoRD[461] 取自 SuperGLUE 基准，用于多选阅读理解和基于引用的推理任务。

这些数据集的优势在于其提供了标准化的测试场景和广泛的任务覆盖范围。然而，这类静态数据集的一个主要局限性在于，它们难以反映动态、真实场景中知识的时效性需求。例如，WikiEval 数据集虽然由 RAGAs[462] 基准基于 2022 年后更新的 Wikipedia 页面构建，但仍然无法完全解决动态场景中更新频繁的知识需求挑战。

2. 自动生成的数据集

随着大语言模型的强大能力得以广泛应用，数据集的构造过程得到了显著简化。研究者能够利用大语言模型设计查询及其对应的答案，从而为特定评估目标生成定制化数据集。这种生成方

法的灵活性使得数据集能够更好地适应实际需求，同时对评估 RAG 系统的动态知识处理能力提出了更高要求。

RGB[463]、MultiHop-RAG[464] 和 CRUD-RAG[465] 是基于自动生成数据集的典型案例。这些基准通过在线新闻文章生成数据集，用于测试 RAG 系统在处理真实世界信息时的表现。数据集内容超越了训练数据的覆盖范围，评估系统对于动态、实时信息的适应能力。CDQA[466] 则结合新闻来源的数据生成评估集，并引入标签器辅助构建更复杂的评价任务。DomainRAG[467] 结合了单文档、多文档、单轮对话和多轮对话等多种任务类型，数据集内容基于高校招生和注册信息的年度变化生成。该数据集通过提供更新后的信息，强迫系统利用动态的知识库来完成任务，对 RAG 系统的时效性和适应性进行了全面评估。OmniEval[468] 提出一个专门针对金融领域的自动和全方位的 RAG 评估基准，将查询分为抽取式问答、多跳推理、对比、对话和长文本问答 5 个任务类别以及 16 个金融主题，如股票市场、投资银行、财产保险等，形成 RAG 场景矩阵，实现对多样查询场景的结构化评估。OmniEval 包含 11400 个自动生成的测试示例和 1700 个人工标注的测试示例。

3. 数据集的构建策略

在应用场景中，评估数据集的构建需要结合任务特点与评估目标，以全面衡量 RAG 系统的性能。为了测试 RAG 系统在动态真实场景中的表现，部分评估基准（如 RGB、MultiHop-RAG、CRUD-RAG 和 DomainRAG）通过爬取新闻、年度变化数据或实时信息生成评估数据集。这些动态数据集能够有效检验系统在面对训练数据未覆盖的最新信息时的适应能力和处理效率，从而评估其动态性和时效性。

针对特定评估目标，定制化数据集的构建能够更好地模拟复杂任务场景。例如，DomainRAG 设计了结合单轮与多轮对话的任务，测试系统在复杂用户交互中的表现；CDQA 则通过多文档生成任务，评估系统在整合和分析多源信息时的能力。通过任务定制化，评估数据集能够更精确地反映系统在特定场景下的实际性能。

数据集的多样性和覆盖率是评估 RAG 系统全面性能的关键指标。通过结合多种任务类型（如开放域问答、多轮对话、事实验证等）和多样化的数据来源（如新闻、百科全书、结构化数据库等），评估数据集能够更全面地展现系统在不同领域中的适应性和泛化能力。这种多样化设计确保了 RAG 系统在广泛应用场景中的可靠性和实用性。

9.5.4 评估指标

评估指标是衡量 RAG 系统性能的核心工具，直接影响评估结果的可信度和系统优化的方向。在评估 RAG 系统时，需要对各种评估指标有深入的理解，以便准确衡量评估目标。由于 RAG 系统由检索组件和生成组件构成，以下从检索模块、生成模块和整体系统三个层面展开详细讨论。

1. 检索模块的评估指标

检索模块的评估指标需要全面反映系统在复杂信息环境中的表现，不仅关注检索结果的相关性和准确性，还需涵盖多样性与鲁棒性，以衡量系统在动态、海量且可能包含误导性信息的知识库中的适应能力。针对检索模块的评估指标可以分为基于排序和非基于排序两大类。

基于排序的指标评估相关项目在排序列表中的呈现顺序，重视相关项目在列表中的排名位置。评测指标主要有：

平均倒数排名（Mean Reciprocal Rank, MRR）是一组查询中，第一个正确答案的倒数排名的平均值，公式为：

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i} \tag{9.16}$$

其中 $|Q|$ 是查询的数量，$ranl_i$ 是第 i 个查询的第一个相关文档的排名位置。

平均准确率均值（Mean Average Precision, MAP）是每个查询的平均准确率得分的平均值，公式为:

$$MAP = \frac{1}{|Q|}\sum_{q=1}^{|Q|}\frac{\sum_{k=1}^{n}(P(k)\times rel(k))}{|\text{第 q 个查询的相关文档数量}|} \tag{9.17}$$

其中 $P(k)$ 是指在排名列表中截止到 k 位置时的精确率，$rel(k)$ 是一个指示函数，当排名为 k 的项目是相关文档时，其值为 1，否则为 0，n 是检索到的文档数量。

非基于排名的指标通常用于评估二元结果，即一个项目是否相关，而不考虑该项目在排序列表中的位置。需要注意的是，以下公式只是这些指标的一种形式，每个指标的定义可能因评估任务的不同而有所差异。评测指标主要有：

准确率（Accuracy）是指在检查的所有案例中，真实结果（包括真阳性和真阴性）所占的比例。

精确率（Precision）是检索到的实例中相关实例的比例，公式为：

$$\text{Precision} = \frac{TP}{TP+FP} \tag{9.18}$$

其中 TP 表示真阳性，FP 表示假阳性。

召回率 @k（Recall@k）是在仅考虑前 k 个结果的情况下，已检索到的相关实例占总相关实例的比例，公式为：

$$\text{Recall@}k = \frac{|RD\cap Top_{kd}|}{|RD|} \tag{9.19}$$

其中 RD 是指真正相关的文档集合，Top_{kd} 是指检索到的前 k 个文档。

2. 生成模块的评估指标

生成组件负责利用检索结果，根据用户查询生成连贯、准确的回答。其性能评估需要全面衡量生成内容的质量、真实性以及与检索内容的一致性。以下从准确性与忠实度、连贯性与流畅性、生成内容的多维度质量、开放性任务的多样性以及真实性检测五个方面详细说明生成组件的评估指标。

（1）准确性与忠实度（Accuracy and Faithfulness）：生成内容必须忠实于检索结果，并准确回答用户的查询。以下是常用的评估指标：

BLEU（Bilingual Evaluation Understudy）通过计算生成内容与参考答案之间的 n-gram 重叠程度来评估生成的准确性。计算公式如下：

$$BLEU = BP \times \exp\left(\sum (w_n \times \log(p_n))\right) \tag{9.20}$$

其中，BP 为长度惩罚因子，防止生成内容过短，w_n 表示 n-gram 的权重，p_n 表示生成文本与参考文本中 n-gram 的匹配概率。BLEU 适合评估结构化任务（如机器翻译），但对开放性任务的灵活性有限。

ROUGE（Recall-Oriented Understudy for Gisting Evaluation）ROUGE 主要用于评估生成摘要任务，衡量生成内容与参考答案的文本片段重叠程度。常用的 ROUGE 指标有 ROUGE-N（基于 n-gram）、ROUGE-L（基于最长公共子序列，LCS）。计算公式如下：

$$\text{ROUGE-N} = \frac{\sum(\text{Overlapping N-grams})}{\sum(\text{Reference N-grams})} \tag{9.21}$$

$$\text{ROUGE-L} = F_1(\text{LCS}) = \frac{(1+\beta^2) \times \text{Precision}_{\text{LCS}} \times \text{Recall}_{\text{LCS}}}{(\beta^2 \times \text{Precision}_{\text{LCS}} + \text{Recall}_{\text{LCS}})} \tag{9.22}$$

其中，$\text{Precision}_{\text{LCS}}$ 表示生成文本中最长公共子序列的精确率，$\text{Recall}_{\text{LCS}}$ 表示参考文本中最长公共子序列的召回率。

Exact Match（EM）用于评估生成回答与参考答案的完全一致性，常用于问答任务，公式如下所示：

$$\text{EM} = \frac{\text{正确答案数量}}{\text{全部答案数量}} \tag{9.23}$$

EM 适合有明确标准答案的任务，对开放性生成任务不适用。

（2）连贯性与流畅性（Coherence and Fluency）：生成内容需要逻辑连贯、语法正确且自然流畅。这些评估通常通过人工评分或基于语言模型的自动评分完成。

人工评分是常用方法之一，通过评分标准量化生成内容的表现。例如，流畅性评分从 0（完全

不流畅）到 5（极其流畅），连贯性评分从 0（完全不连贯）到 5（逻辑严谨且连贯），以此反映生成文本在语言表达上的自然程度和逻辑性。此外，人工评分能够结合具体情境进行主观判断，适用于需要细腻评估的场景。

另一种方法是基于语言模型的自动评分，通过计算生成文本的条件概率来评估其语言质量和逻辑性。这种方法利用公式 Fluency Score $= \log P(\text{Generated Text}|\text{Context})$ 量化生成文本在上下文中的自然程度。语言模型评分具有高效性和一致性，尤其适合大规模评估任务，同时可以减少人工评估的成本。这种自动化的方式为连贯性与流畅性评估提供了数据驱动的支持，成为生成模块评估的重要补充手段。

（3）生成内容的多维度质量（Multi-Dimensional Quality）：生成内容的质量需要从多个维度衡量，包括可读性、丰富性和结构化程度。

在可读性方面，可以使用 Flesch Reading Ease[469] 公式计算，该公式是一种广泛使用的英语文本易读性评估工具，由 Rudolph Flesch 在 1948 年提出，具体计算公式如下：

$$\text{RE} = 206.835 - 1.015 * \text{ASL} - 84.6 * \text{ASW} \tag{9.24}$$

其中，RE 表示易读性分数，ASL 是平均句子长度，即单词数除以句子数，ASW 是每个单词的平均音节数，即音节数除以单词数。分值越高，文本越容易阅读。

结构化程度则关注生成内容是否符合特定任务的格式要求。例如，在表格生成任务中，系统需要确保正确生成表头并填充对应数据，以便生成的内容具有清晰的逻辑和易于阅读的展示形式。对于生成摘要任务，结构化程度还可能包括段落分布是否合理、内容是否按照主题分块等。结构化的内容不仅提升了用户体验，还能提高信息的利用效率。丰富性衡量生成内容的全面性和细节程度，通常通过信息覆盖率（ICR）进行评估，其公式为：

$$\text{ICR} = \frac{\text{生成的文本中包含事实个数}}{\text{参考答案中包含事实个数}} \tag{9.25}$$

这一指标反映了生成内容是否充分涵盖了参考内容中的关键信息，同时避免遗漏重要细节。高丰富性的内容能够为用户提供更全面的信息支持，尤其在复杂任务场景中显得尤为重要。

（4）开放性任务的多样性（Diversity in Open-Ended Tasks）：在没有明确标准答案的开放性任务评估中，多样性是重要考量。此类任务要求生成内容在保持与输入主题一致的同时呈现出显著的多样性和创新性。评估生成内容的多样性需要从语义差异性和冗余度两个方面入手，以全面衡量生成内容的丰富程度和信息分布特性。

语义多样性得分（Semantic Diversity Score, SDS）是衡量生成内容语义层面差异性的重要指标，其公式为：

$$\text{SDS} = 1 - \cos(\text{Embedding}_1, \text{Embedding}_2) \tag{9.26}$$

其中 Embedding_1 和 Embedding_2 表示生成内容不同部分的语义嵌入。该指标通过评估生成内容中各部分的语义相似性来计算其多样性，分值越高，表明生成内容在语义表达上越具有差异性，从而更具创造性和多样性。例如，在生成一篇长篇文章时，SDS 可以衡量不同段落之间的思想深度和内容差异，确保生成文本不只是重复或简单扩展输入，而是提供了新颖且多样化的语义表达。

冗余度分析（Redundancy Analysis）则检测生成内容中重复信息的比例，其公式为：

$$\text{Redundancy} = \frac{\text{重复单词或短语个数}}{\text{单词或短语总数}} \tag{9.27}$$

高冗余度表明内容中存在大量重复，缺乏创造性，而低冗余度则意味着生成内容更丰富多样，信息表达更加新颖。在开放性生成任务中，冗余度分析对于避免内容冗长和信息重复至关重要。例如，在创造性写作任务中，低冗余度的文本更能保持读者的兴趣，同时避免单调和无意义的重复。

（5）真实性检测（Hallucination Suppression）：真实性检测在避免生成“幻觉”内容方面发挥关键作用。这类内容对用户体验和系统可靠性具有严重影响，因此需要通过科学的指标进行评估。

FEVER 评分是一种常用的方法，用于测试生成内容与事实的匹配程度。其核心指标是证据支持率（Evidence Support Rate, ESR），其公式为：

$$\text{ESR} = \frac{\text{正确的信息数}}{\text{全部信息数}} \tag{9.28}$$

高 ESR 表明生成的文本有较高比例能够得到检索证据的支持，有助于评估生成内容的事实基础。

误导率（Misleading Rate, MR）是衡量生成内容中包含误导性信息比例的关键指标，其公式为：

$$\text{MR} = \frac{\text{误导性信息数}}{\text{全部信息数}} \tag{9.29}$$

误导性信息是用户最难以察觉的错误类型，因为它通常以真伪混杂的方式呈现。较低的 MR 值意味着生成模型更具可信度，能够生成更可靠的内容。在实际场景中，例如医疗或法律领域，误导性信息可能会导致严重的后果，因此通过 MR 指标能够有效衡量和优化生成模块在这些高敏感性领域的表现。

错误检测率（Error Detection Rate, EDR）则反映生成系统发现并标记错误信息的能力，其公式为：

$$\text{EDR} = \frac{\text{检测出的错误数}}{\text{总错误数}} \tag{9.30}$$

EDR 的高低直接决定了系统在生成内容后续处理中的能力，尤其是在生成内容需要进一步验证或提供错误提示的场景中。例如，在生成开放性回答时，系统需要对可能的错误进行标记或提示，以避免用户直接采信错误信息。这种对错误的主动识别能力不仅提高了生成模块的智能性，还增强

了用户对系统的信任度。

通过综合 FEVER 评分、误导率和错误检测率，可以多维度评估生成模块的真实性检测性能，为生成内容的可靠性提供全面保障。

9.6 RAG 实践

在此前的章节中，已经详细介绍了检索增强生成的基础概念、系统模块构成以及优化训练方法，为深入理解 RAG 技术奠定了理论基础。本节将介绍如何使用 LangChain 框架实现检索增强生成系统。

9.6.1 基础 RAG 系统

使用 LangChain 可以快速构建一个基础的 RAG 系统:

```
# 导入需要的模块和类
import bs4
from langchain import hub
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.document_loaders import WebBaseLoader
from langchain_community.vectorstores import Chroma
from langchain_community.embeddings import HuggingFaceBgeEmbeddings
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnablePassthrough
from langchain_ollama.llms import OllamaLLM

#### 索引 ####
# 1. 从指定目录中读取所有文件的数据
# 使用目录读取器 SimpleDirectoryReader 加载数据
docs = SimpleDirectoryReader("./RAGDoc").load_data()

# 2. 文件分割，采用滑动窗口方法进行分块，分块大小为 1000，块之间重叠为 200
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
splits = text_splitter.split_documents(docs)

# 3. 文本嵌入表示模型初始化
embed_model = HuggingFaceBgeEmbeddings(model_name= "BAAI/bge-large-zh-v1.5")

# 4. 使用 Chroma 构建向量检索
vectorstore = Chroma.from_documents(documents=splits, embedding=embed_model)
retriever = vectorstore.as_retriever()

#### 检索 和 生成 ####
# 3. 构建 Prompt 模板，使用现有的 rlm/rag-prompt
prompt = hub.pull("rlm/rag-prompt")

# 4. 使用 Ollama 接入本地大语言模型
llm = OllamaLLM(model="qwen2.5")

# 5. 检索后优化
def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)

# 6. 构建 RAG 链
rag_chain = (
    "context": retriever | format_docs, "question": RunnablePassthrough()
    | prompt
    | llm
    | StrOutputParser()
)

# 7. 使用 Rag 链进行查询
rag_chain.invoke(" 复旦大学有几个校区?")

# 8. 打印从查询引擎返回的响应
print(response)
```

9.6.2 查询分解与检索结果融合 RAG 系统

针对复杂问题，RAG 系统在处理查询之前的优化阶段，通常需要引入查询分解等技术。这是因为复杂查询往往包含多个子问题或逻辑层次，直接检索可能难以获得高质量的结果。通过查询分解，可以将复杂查询拆分成更小、更易处理的子查询，从而提高检索的准确性和生成回答的质量。如图9.20所示，查询分解作为预处理步骤加入基础 RAG 系统中。

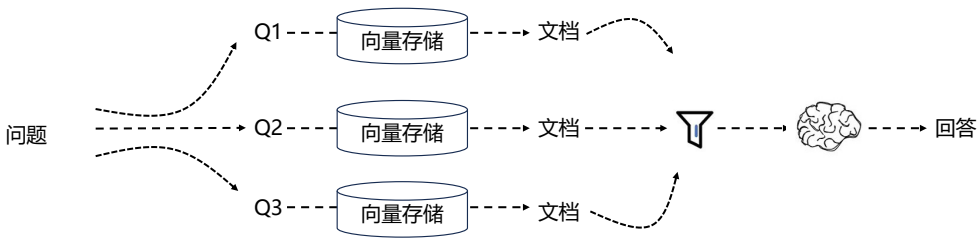

图 9.20　包含查询分解与检索结果融合的 RAG 系统

使用 LangChain 可以快速构建一个包含查询分解与检索结果融合的 RAG 系统:

```
# 导入需要的模块和类
import bs4
from langchain import hub
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.document_loaders import WebBaseLoader
from langchain_community.vectorstores import Chroma
from langchain_community.embeddings import HuggingFaceBgeEmbeddings
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnablePassthrough
from langchain_ollama.llms import OllamaLLM
from langchain.prompts import ChatPromptTemplate
from langchain.load import dumps, loads
from langchain_core.runnables  import RunnablePassthrough

#### 索引 ####
docs = SimpleDirectoryReader("./RAGDoc").load_data()
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
splits = text_splitter.split_documents(docs)
embed_model = HuggingFaceEmbedding(model_name= "BAAI/bge-large-zh-v1.5")
vectorstore = Chroma.from_documents(documents=splits, embedding=embed_model)
retriever = vectorstore.as_retriever()

# 使用 Ollama 接入本地大语言模型
llm = OllamaLLM(model="qwen2.5")

# 构造 query 分解 Prompt
template = """You are a helpful assistant that generates multiple search queries based on
                a single input query. \n
                Generate multiple search queries related to: question \n
                Output (4 queries):"""
prompt_rag_fusion = ChatPromptTemplate.from_template(template)

# 构造 query 分解链
generate_queries = (
    prompt_rag_fusion
    | llm
    | StrOutputParser()
    | (lambda x: x.split("\n"))
)
```

```
# 定义多查询融合函数
def reciprocal_rank_fusion(results: list[list], k=60):
    """ Reciprocal_rank_fusion that takes multiple lists of ranked documents
        and an optional parameter k used in the RRF formula """

    # 初始化一个字典，用于存储每个文档的融合分数
    fused_scores =

    # 遍历每个文档
    for docs in results:
        # 根据排名遍历列表中的文档排
        for rank, doc in enumerate(docs):
            # 将文档转换为字符串格式，作为键使用（假设文档可以序列化为 JSON）
            doc_str = dumps(doc)
            # 如果文档尚未在融合分数字典 fused_scores 中，则添加它，初始分数为 0
            if doc_str not in fused_scores:
                fused_scores[doc_str] = 0
            # 如果文档已存在，则检索其当前分数
            previous_score = fused_scores[doc_str]
            # 使用 RRF : 1 / (rank + k) 公式 更新文档分数
            fused_scores[doc_str] += 1 / (rank + k)

    # 根据融合分数对文档进行排序，以获取最终的重排序结果
    reranked_results = [
        (loads(doc), score)
        for doc, score in sorted(fused_scores.items(), key=lambda x: x[1], reverse=True)
    ]

    # 将重排序结果作为包含文档和融合分数的元组列表返回
    return reranked_results

question =" 复旦大学有几个校区?"

# 构建查询融合链
retrieval_chain_rag_fusion = generate_queries | retriever.map() | reciprocal_rank_fusion
docs = retrieval_chain_rag_fusion.invoke("question": question)
print(len(docs))

# 构建包含查询分解的 RAG 链
template = """Answer the following question based on this context:
{context}
Question: {question}
"""

prompt = ChatPromptTemplate.from_template(template)
final_rag_chain = (
    "context": retrieval_chain_rag_fusion,     "question": itemgetter("question")
    | prompt
    | llm
    | StrOutputParser()
)

print(final_rag_chain.invoke("question":question))
```

10. 大语言模型效率优化

大语言模型在自然语言理解与生成等任务中展现了卓越的能力，不仅推动了人工智能技术的快速发展，也为社会各领域的应用带来了深远的影响。然而，这些强大的能力背后伴随着巨大的资源消耗，包括计算、存储和能源需求，这对环境、经济以及技术可持续性带来了严峻挑战。因此，如何在保持模型性能的同时提高其效率，已成为当前大模型研究中的重要议题。为应对这一问题，研究者们从模型、数据和计算框架等多个角度探索了提升大模型效率的优化方法。通过模型压缩、量化、数据选择和优化训练框架等技术，显著降低了训练与推理成本，为实现更加可持续和普惠的人工智能提供了可能性。

本章将重点从模型、训练和推理三个角度系统性探讨提升大语言模型效率的技术进展，分别涵盖模型压缩与优化、训练效率优化，以及推理效率优化和专用框架的设计与应用。

10.1 效率优化基础

大语言模型的推理过程遵循自回归模式（Autoregressive Pattern），如图10.1 所示。例如，针对输入“复旦大学位”，模型预测“于”的概率比“置”的概率高。因此，在第一次迭代后，“于”字被附加到原始输入中，并将“复旦大学位于”作为一个新的整体输入模型以生成下一个词元。这个生成过程持续进行，直到生成表示序列结束的 <eos> 标志或达到预定义的最大输出长度为止。大语言模型的推理过程与其他深度学习模型（如 BERT、ResNet 等）非常不同，BERT 的执行时间通常是确定且高度可预测的。但是，在大语言模型的推理过程中，虽然每次迭代执行的时间仍然具有确定性，但迭代次数（输出长度）是未知的，这使得一个大语言模型推理任务的总执行时间是不可预测的。

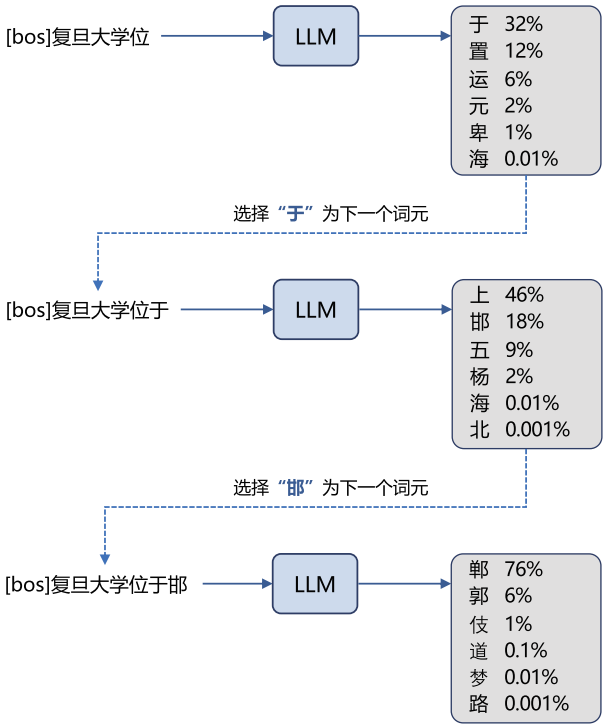

图 10.1 大语言模型推理遵循自回归模式

在每次 GPT 推理中，对每个词元的自注意力操作需要其前面词元的键和值。最简单且无状态的实现需要在每次迭代中重新计算所有的键和值，这会导致大量额外的计算开销。为了避免这种重新计算的开销，FAIRSEQ[470] 提出了键值缓存（Key-Value Cache），即在迭代中保存键和值，以便重复使用。根据上述方法和技术，大语言模型的推理过程可以分为预填充阶段（Prefilling Stage）和解码阶段（Decoding Stage）两个阶段，如图10.2所示。在预填充阶段，模型会计算并存储初始输入词元的键-值（KV）缓存，同时生成第一个输出词元。随后进入解码阶段，模型逐个生成后续输出词元，并在每一步更新 KV 缓存，直至完成整个推理过程。

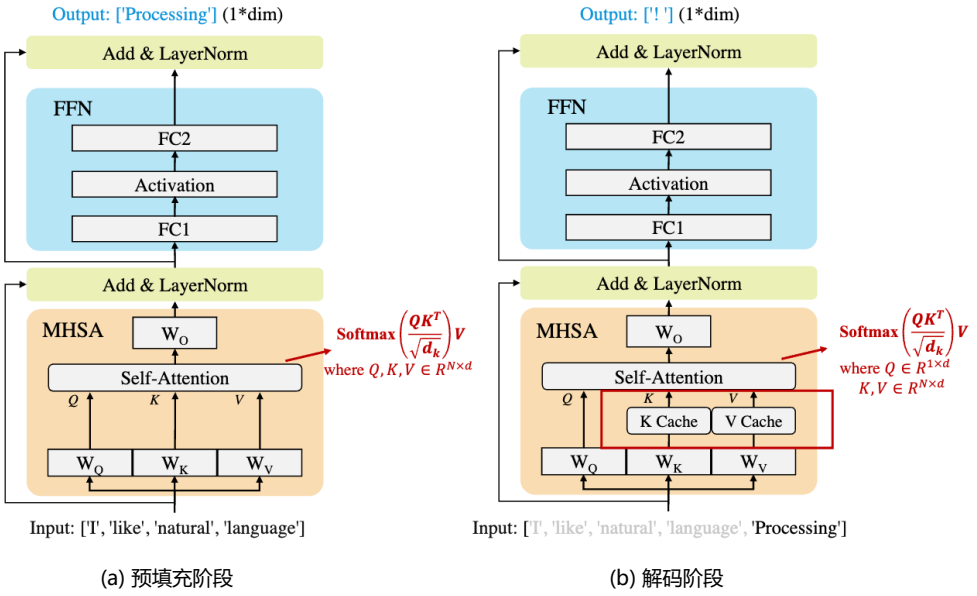

图 10.2 大语言模型解码两阶段[471]

键值缓存在不同阶段的使用方式如图10.3 所示。在预填充阶段，即第一次迭代中，将输入的提示词进行处理，为大语言模型的每个 Transformer 层生成键值缓存。在解码阶段，大语言模型只需要计算新生成词元的查询、键和值。利用并更新键值缓存，逐步生成后面的词元。

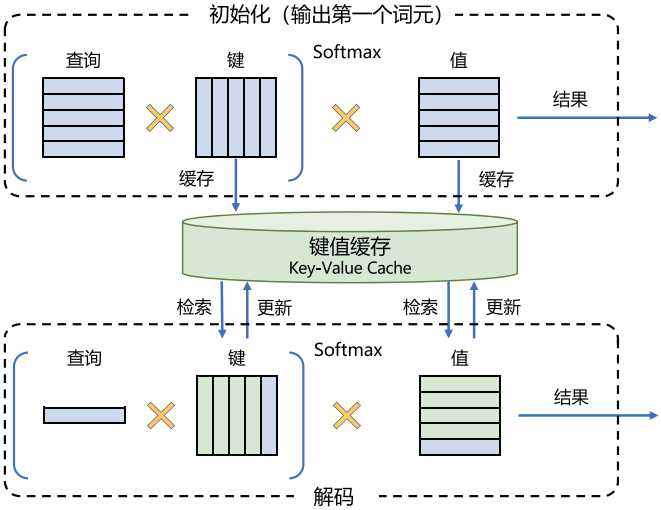

图 10.3 键值缓存在不同阶段的使用方式[472]

在资源受限的环境中部署大语言模型，同时保持其强大的性能，是当前实践者和研究人员面临的核心难题。例如，部署一个拥有 700 亿参数的 LLaMA-2-70B 模型，需要克服存储和计算资源的多重限制。该模型的权重以 FP16 格式存储时占用约 140 GB 显存，这意味着至少需要 6 张 RTX 3090 Ti GPU（每张显存 24 GB）或 2 张 NVIDIA A100 GPU（每张显存 80 GB）才能满足推理需求。此外，在 2 张 NVIDIA A100 GPU 上生成单个输出词元（Token）的时间约为 100 毫秒，因此生成一个包含数百个词元的序列可能耗时超过 10 秒。除了存储需求和延迟问题，推理过程还需综合考虑吞吐量、能耗和功耗等关键效率指标，以实现更高效的资源利用。

在大语言模型的推理过程中，效率指标主要受到三个关键因素的影响：计算成本、内存访问成本和内存使用情况。文献 [473] 提出了基于 Roofline 模型的系统化分析，深入探讨了这些因素如何限制推理效率。以下将进一步分析导致大语言模型推理效率低下的三大核心原因，分别是模型规模、自注意力机制和解码方法。

1. 模型规模的影响：主流的大型语言模型通常包含数十亿到数万亿的参数。例如，LLaMA-70B 拥有 700 亿参数，而 GPT-3 的规模更是高达 1750 亿参数。这类超大规模模型显著增加了推理过程中的计算成本、内存访问成本和内存使用量。随着模型参数规模的增大，推理所需的计算资源和显存容量也随之增加。同时，模型权重需要频繁从高带宽内存（HBM）加载到 GPU 芯片，这不仅加剧了内存访问延迟，还显著提高了能耗。此外，大规模模型的权重存储和处理会占用大量显存资源，从而降低整体的资源利用效率。

2. 自注意力操作的影响：在推理过程中，自注意力机制是计算复杂度的主要来源之一。正如前文所述，在预填充阶段，自注意力操作的计算复杂度随着输入长度的增加呈现出二次增长（$O(n^2)$）。这意味着，当输入长度较长时，自注意力机制会显著增加计算成本、内存访问成本和内存使用量。例如，在处理长文本时，模型需要为每个词元计算注意力权重矩阵，这不仅显著加重了计算负担，还导致显存占用大幅上升。因此，自注意力机制的高复杂度成为推理效率低下的关键瓶颈之一。

3. 解码方法的影响：大语言模型通常通过自回归解码方法逐步生成输出词元。在解码的每一步，模型需要将全部权重从高带宽内存（HBM）加载到 GPU 芯片上，这大幅增加了内存访问成本。此外，随着输入长度的增长，键值缓存（KV 缓存）的大小也会不断扩大。这不仅消耗了大量显存资源，还可能引发内存碎片化和不规则的内存访问模式，进一步降低推理效率。特别是在生成长序列时，KV 缓存的管理成为影响推理性能的关键因素之一。

为了更清晰地了解大语言模型推理过程中的关键效率指标，图10.4 直观地展示了推理延迟和内存使用的相关情况。**首词元延迟**（First Token Latency）指的是在预填充阶段生成首个输出词元所需的时间。**输出词元间**（Per-output Token Latency）描述了解码阶段中生成单个输出词元的平均耗时。**生成延迟**（Generation Latency）则衡量了生成整个输出序列的总时间。在模型的内存使用方面，**模型大小**（Model Size）表示存储模型权重所需的内存空间，**KV 缓存大小**（KV Cache Size）则指存储键值缓存所需的内存。两者共同决定了推理过程中**峰值内存**（Peak Memory）的需求，而峰值内存通常接近模型权重和 KV 缓存所需内存的总和。除了延迟和内存，吞吐量也是衡量大语

言模型服务性能的重要指标之一。具体来说，**词元吞吐量**（Token Throughput）表示每秒生成的词元数量，而**请求吞吐量**（Request Throughput）则表示服务系统每秒能够完成的请求数量。这些指标共同反映了模型在推理过程中的效率和服务能力。

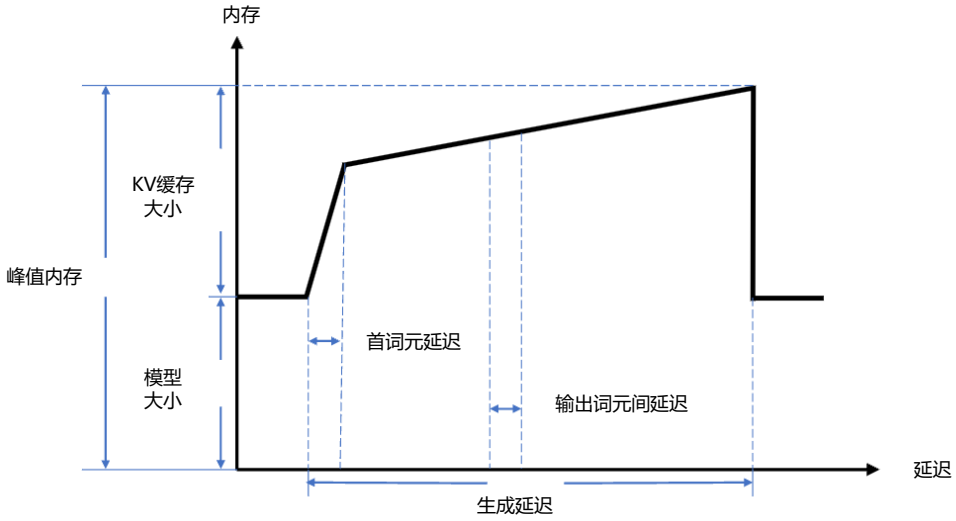

图 10.4 大语言模型解码阶段内存变化[471]

在生成序列的过程中，内存使用量和延迟时间会随着生成词元数量的增加而显著变化。前向传播计算过程中，前一层的输出就是后一层的输入，相邻两层的中间结果也是需要 GPU 显存来保存，中间结果变量也叫激活内存，值相对很小。图10.4 忽略了激活值的大小，但仍然可以清楚地看到，推理过程中的计算和内存需求会随着时间线性或非线性地增加。

为了进一步优化推理效率，需要从以下几个方面入手：一是通过模型压缩技术（如量化、剪枝）来减少模型规模；二是设计更高效的自注意力机制（如稀疏注意力）；三是改进解码方法（如批量解码或并行解码）以降低内存访问成本。这些优化策略将在后续章节中进一步探讨。

10.2 模型优化

模型优化是提升大语言模型推理效率的重要手段，主要集中在优化模型结构和模型压缩两方面。模型结构优化通过设计高效的模型结构直接提升效率，包括高效 FFN 设计、注意力机制优化、MoE 架构设计、Transformer 代替架构设计等，这些内容大部分都在本书第二章大语言模型基础部分进行了介绍。模型压缩则涵盖了多种技术，旨在通过修改模型的数据表示（例如量化）、改变其架构（例如稀疏化、结构优化等）或者知识蒸馏来提高预训练模型的推理效率。

在本节中将着重介绍模型优化中的 Transformer 代替架构、模型量化、模型稀疏化以及模型蒸馏。

10.2.1 Transformer 代替架构

状态空间模型（State Space Model，SSM）是当前研究替代 Transformer 架构的热门方向之一。例如，Mamba[474] 和 Vision Mamba[475] 就是典型的状态空间模型，并在某些自然语言处理和计算机视觉任务中取得了优异的表现。与基于注意力机制的 Transformer 不同，SSM 在计算和存储方面对输入序列长度呈线性复杂度。这种特性显著提升了其在处理长文本序列时的效率，使其成为探索高效架构的重要候选之一。

状态空间模型假设动态系统可以通过其在某一时刻（时间 t）的状态来进行预测。这个预测过程通常基于两个核心方程：第一个方程描述系统状态随时间的变化（即系统的动力学特性），第二个方程将系统的状态映射到可观测值或输出。这种建模方式使 SSM 能够精确捕捉系统的动态行为，并利用当前状态对未来的状态或输出进行预测。两个方程可以如下形式化表示：

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t) \tag{10.1}$$

$$y(t) = \boldsymbol{C}h(t) + \boldsymbol{D}x(t) \tag{10.2}$$

其中，$\boldsymbol{A}$ 是状态转移矩阵、$\boldsymbol{B}$ 表示控制量对状态量的影响、$\boldsymbol{C}$ 表示当前状态量对输出影响和 $\boldsymbol{D}$ 表示当前控制量对输出影响，上述四个矩阵都是可学习的，也称为模型参数，h 表示中间状态，x 表示输入序列。

状态空间模型的基本过程如图10.5所示。输入信号 x 与矩阵 $\mathbf{B}$ 相乘，生成一个向量，用于表示输入 x 对系统状态的影响。状态表示（State Representation）h 是一个隐向量，包含了系统的核心“知识”。通过与矩阵 $\mathbf{A}$ 相乘，状态表示描述了内部状态之间的关联，从而体现系统的动态特性。在预测输出之前，需要根据当前状态和输入信号更新状态。最后，通过矩阵 $\mathbf{C}$ 将状态映射到输出空间，利用矩阵 $\mathbf{D}$ 提供从输入到输出的直接信号（通常被称为跳跃连接（Skip Connection）），生成最终的输出。矩阵 $\mathbf{C}$ 描述了状态与输出之间的关系，即如何将状态转换为输出结果。

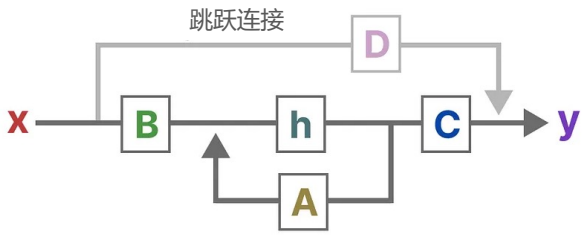

图 10.5　状态空间模型基本架构

为了使 SSM 模型适应离散输入（如文本序列），可以采用零阶保持（Zero-Order Hold, ZOH）技术。其原理是每次接收到一个离散信号时，保持该信号的值，直到下一个新的离散信号到达为止。通过这种方法，离散输入信号被转换为连续信号，从而使状态空间模型能够更高效地处理和计算。这种方式使 SSM 能够在离散输入序列的基础上生成连续的状态表示。保持该值的时间长短由一个可学习参数表示，称为步长 Δ，表示输入的分辨率。离散化 SSM 允许以特定的时间步长而不是连续信号来制定问题。将当前控制量对输出的影响 $\mathbf{D}$ 忽略，离散化 SSM 可以如下形式化表示：

$$h_t = \bar{\boldsymbol{A}} h_{t-1} + \bar{\boldsymbol{B}} x_t \tag{10.3}$$

$$y_t = \boldsymbol{C} h_t + \boldsymbol{D} x_t \tag{10.4}$$

$$\bar{\boldsymbol{A}} = e^{\Delta \boldsymbol{A}} \tag{10.5}$$

$$\bar{\boldsymbol{B}} = \left(e^{\Delta \boldsymbol{A}} - I\right) \boldsymbol{A}^{-1} \boldsymbol{B} \tag{10.6}$$

离散化 SSM 的序列化表示结构与循环神经网络（RNN）类似。但是与 RNN 不同的是，离散化 SSM 在计算输出 y_t 时，采用了线性变换，而没有使用激活函数进行非线性化。这一改变使得可以将 SSM 表示为卷积形式的状态预测，能够像卷积神经网络（CNN）一样实现并行训练。这使得 SSM 在处理大规模数据时具有较高的计算效率。

Mamba 模型[474] 采用了离散化的状态空间模型，并引入了一种改进的选择机制，称为选择性状态空间模型（Selective State Space Models）。这一机制使模型能够根据输入内容有选择地传播或遗忘信息，从而增强了表达能力。为了确保选择机制的 SSM 能在硬件上高效运行，Mamba 设计了一种结合内核优化与重新计算的硬件感知算法，有效避免了中间状态的存储，大幅提升了速度和内存效率。此外，Mamba 将 H3[476] 中的 SSM 块与 Transformer 中的 MLP 块整合为一个简化的模块，并通过重复堆叠这些模块构建整体架构。这一简化设计进一步提升了训练和推理效率。

Mamba 的网络结构对 GPU 的计算高度友好，尤其在数据交互方面展现了卓越的性能。其数据交互主要集中在 GPU 与片上 SRAM 之间，这种交互完全发生在 GPU 芯片内部，具有极高的速度，显著提升了数据访问和处理效率。在性能表现上，Mamba 在推理速度和准确性方面均表现优异。得益于其结构设计能够更有效地利用更长的上下文，Mamba 在 DNA 和音频建模任务中表现出色，并在依赖远程关系的复杂任务上超越了此前的模型。

在此基础上，一些后续工作进一步改进了 Mamba 模型的架构，推动了状态空间模型的发展与应用。MambaFormer[477] 将标准 Transformer 与 SSM 模型相结合，通过用 SSM 层替代 Transformer 中的前馈神经网络（FFN）层，实现了两种架构的融合。这种设计充分利用了 Transformer 在捕捉局部特征上的优势，同时借助 SSM 的长距离建模能力，使模型在处理复杂任务时表现得更加高效和精准。DenseMamba[478] 针对传统 SSM 中隐藏状态容易退化的问题进行了深入研究。为了缓解隐藏状态在深层网络中逐渐丢失信息的问题，DenseMamba 在 SSM 架构中引入了密集连接