

6 检索增强生成

在海量训练数据和模型参数的双重作用下，大语言模型展示出了令人惊艳的生成能力。然而，由于训练数据的正确性、时效性和完备性可能存在不足，其难以完全覆盖用户的需求；并且，根据“没有免费午餐”[55]定理，由于参数空间有限，大语言模型对训练数据的学习也难以达到完美。上述训练数据和参数学习上的不足将导致：大语言模型在面对某些问题时无法给出正确答案，甚至出现“幻觉”，即生成看似合理实则逻辑混乱或违背事实的回答。为了解决这些问题并进一步提升大语言模型的生成质量，我们可以将相关信息存储在外部数据库中，供大语言模型进行检索和调用。这种从外部数据库中检索出相关信息来辅助改善大语言模型生成质量的系统被称之为检索增强生成（Retrieval-Augmented Generation, RAG）。本章将介绍 RAG 系统的相关背景、定义以及基本组成，详细介绍 RAG 系统的常见架构，讨论 RAG 系统中知识检索与生成增强部分的技术细节，并介绍 RAG 系统的应用与前景。

* 本书持续更新，GIT Hub 链接为：<https://github.com/ZJU-LLMs/Foundations-of-LLMs>。

6.1 检索增强生成简介

检索增强生成（RAG）旨在通过检索和整合外部知识来增强大语言模型生成文本的准确性和丰富性，其是一个集成了外部知识库、信息检索器、大语言模型等多个功能模块的系统。RAG 利用信息检索、深度学习等多种技术为大语言模型在生成过程中引入最新的、特定领域的知识，从而克服传统大语言模型的局限性，提供更加精准和可靠的生成内容。本节我们将主要介绍 RAG 系统的相关背景、定义以及基本组成。

6.1.1 检索增强生成的背景

大语言模型在多种生成任务上展现出了令人惊艳的能力，其可以辅助我们撰写文案、翻译文章、编写代码等。但是，大模型生成的内容可能存在“幻觉”现象——生成内容看似合理但实际上逻辑混乱或与事实相悖。这导致大语言模型生成内容的可靠性下降。“幻觉”现象可能源于大语言模型所采用的训练数据，也可能源于模型本身。

1. 训练数据导致的幻觉

训练数据是大语言模型知识的根本来源。训练数据在采集完成后直接用于训练模型。但是其中包含的知识可能在模型训练后又发生了更新。这将导致**知识过时**的问题。不仅如此，知识在训练数据采集完成后仍会新增，并且训练数据采集也无法覆盖世间所有知识，尤其是垂域知识，这将导致**知识边界**的问题。此外，训练数据中还可能包含不实与偏见信息，从而导致**知识偏差**问题。

对于**知识过时**问题，由于训练数据**涵盖的知识止步于大语言模型训练的时间截面**，其掌握的知识无法与现实世界同步更新。此处，以 ChatGPT 为例¹，对于问

¹本章撰写时，所使用的 ChatGPT 版本的训练数据截止到 2022 年。

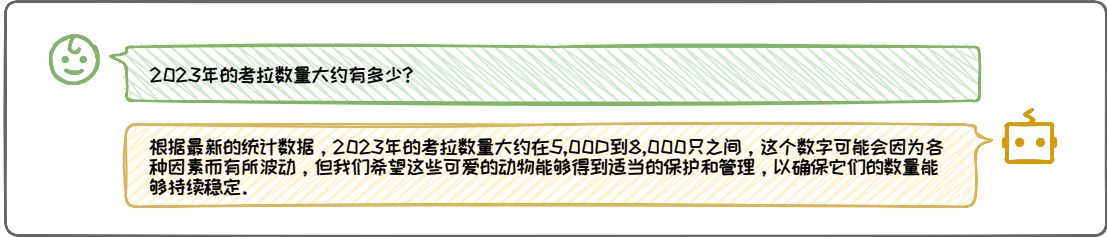


图 6.1: 知识过时引起的幻觉现象示例。

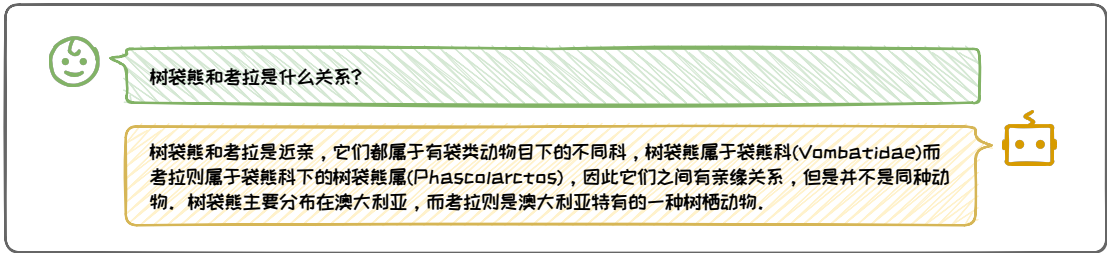


图 6.2: 模型自身导致的幻觉示例。

题“2023 年的考拉数量大约有多少?”, 模型的回答为 5,000 到 8,000 只之间, 如图 6.1 所示。然而, 正确数量应约为 86,000 到 176,000 只。由于 ChatGPT 的完成训练的时间节点是 2022 年, 导致它无法掌握 2023 年的知识, 给出了错误的回答。

虽然大语言模型的训练数据非常庞大, 但仍然是有限的。因此, 模型内部的知识必然存在**知识边界**, 即缺乏某些特定领域的知识。例如, 当我们想知道考拉的基因数量时, 模型可能无法提供正确的信息, 因为预训练数据中并不包含相关信息。此外, 由于大语言模型的语料很多是从互联网直接爬取而未核验的, 其中可能会存在含偏向某些特定观点或存在事实性偏差的低质量数据, 从而带来**知识偏差**, 导致模型输出存在不良偏差。

2. 模型自身导致的幻觉

除了训练数据的影响, 我们还发现在某些场景下, 即使训练数据中已经包含了相关知识, 大语言模型仍然会出现幻觉现象。如图 6.2 中的例子所示, 我们同样使用 ChatGPT 进行测试, 可以看到模型并没有意识到考拉是树袋熊的音译别名, 而错误的把这两个名称认为是两种不同的动物, 偏离了事实。为了进一步探究大语

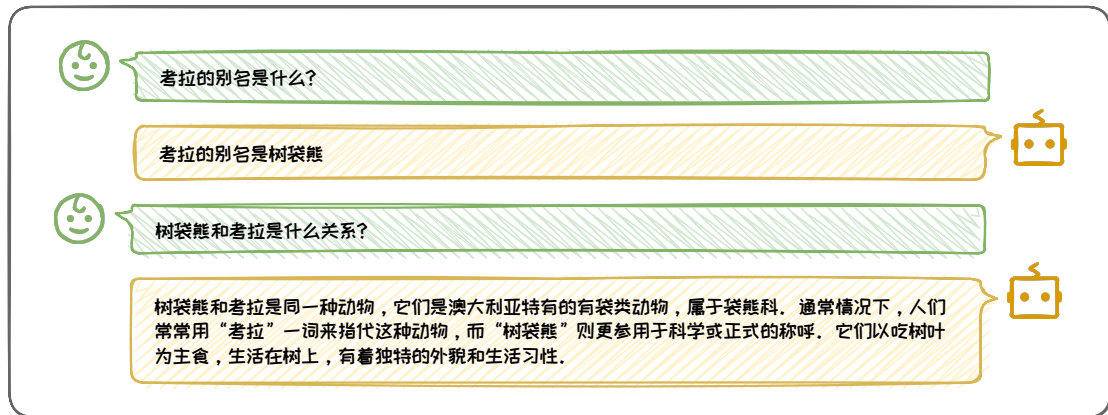


图 6.3: 内部泛化能力不足的进一步示例。

言模型是否真的不具备相关知识。我们进行了进一步实验，结果如图 6.3 所示，此时大语言模型给出了正确回答。上述示例表明，大语言模型实际上包含了问题的正确知识，但依然出现了偏差的回答。

上述偏差可能来自于模型自身，可能的因素包括：(1) **知识长尾**：训练数据中部分信息的出现频率较低，导致模型对这些知识的学习程度较差；(2) **曝光偏差**：由于模型训练与推理任务存在差异，导致模型在实际推理时存在偏差；(3) **对齐不当**：在模型与人类偏好对齐阶段中，偏好数据标注不当可能引入了不良偏好；(4) **解码偏差**：模型解码策略中的随机因素可能影响输出的准确性。

上述幻觉问题极大地影响了大语言模型的生成质量。这些问题的成因主要是大语言模型缺乏相应的知识或生成过程出现了偏差，导致其无法正确回答。借鉴人类的解决方式，当我们遇到无法回答的问题时，通常会借助搜索引擎或查阅书籍资料来获取相关信息，进而帮助我们得出正确答案。自然地，对于大语言模型不熟悉的知识，我们是否也可以查找相关的信息，从而帮助它得到更准确的回答呢？为了验证这一设想，我们可以进行一个简单的实验，同样是上面的两个例子，我们简单地以 Prompt 的形式加入相关的外部知识，如图 6.5、图 6.7 所示，模型很自然

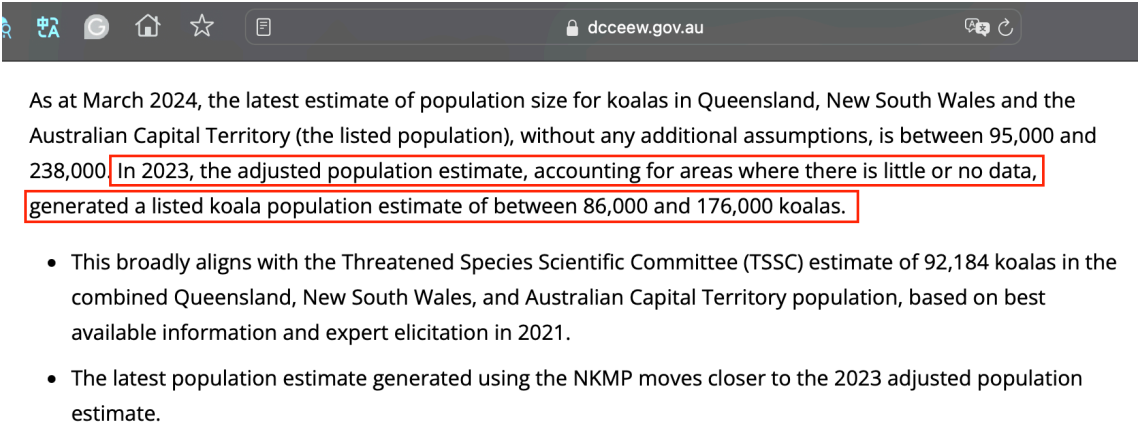


图 6.4: 关于 2023 树袋熊数量的网络资料截图。

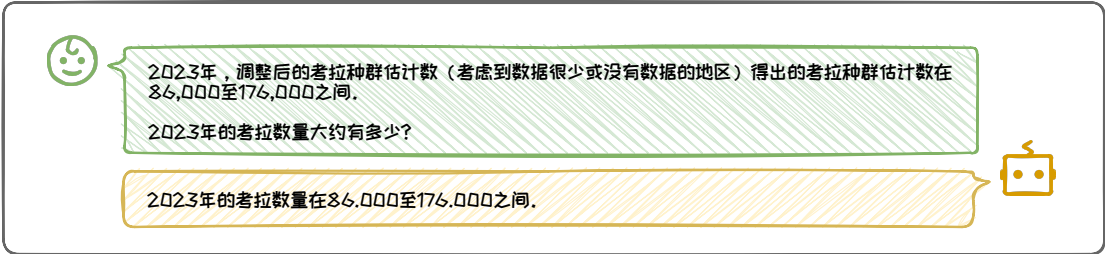


图 6.5: 添加外部信息纠正知识过时引起的幻觉示例。



图 6.6: 关于树袋熊名称的维基百科截图。

地得出了正确的回答，其中图 6.4²、图 6.6³为对应的知识来源。这种思路便是检索增强生成 (Retrieval-Augmented Generation, RAG) 的核心思想。接下来，我们对 RAG 系统进行简要介绍。

²<https://www.dcceew.gov.au/environment/biodiversity/threatened/species/koalas/national-koala-monitoring-program>
³<https://zh.wikipedia.org/wiki/>

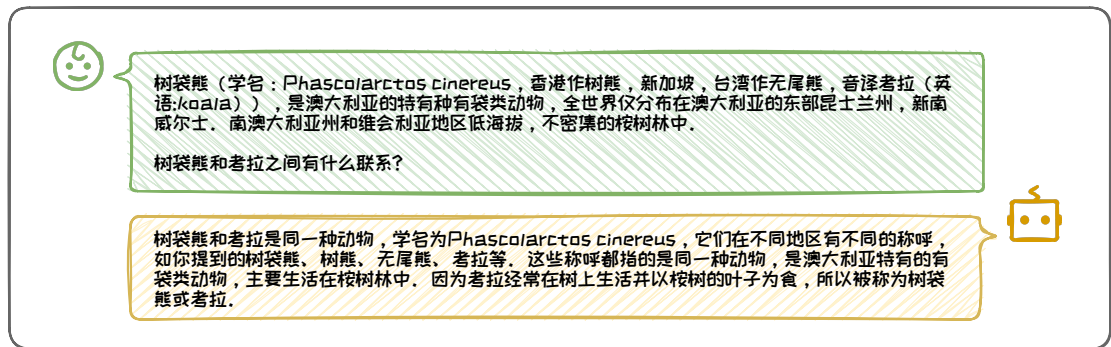


图 6.7: 添加外部信息纠正模型自身引起的幻觉示例。

6.1.2 检索增强生成的组成

RAG 的概念最早出现在 Facebook AI Research 的论文 Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks [28] 中。其通常集成了**外部知识库 (Corpus)**、**信息检索器 (Retriever)**、**生成器 (Generator)**，即大语言模型) 等多个功能模块。通过结合外部知识库和大语言模型的优势，大幅提升模型在开放域问答、多轮对话等任务中的生成质量，其基本架构如图 6.8 所示。具体而言，给定一个**自然语言问题 (Query)**，检索器将问题进行编码，并从知识库（如维基百科）中高效检索出与问题相关的文档。然后，将检索到的知识和原始问题一并传递给大语言模型，大语言模型根据检索到的知识和原始问题生成最终的输出。RAG 的核心优势在于不需要对大语言模型的内部知识进行更新，便可改善大语言模型的幻觉现象，提高生成质量。这可以有效避免内部知识更新带来的计算成本和对旧知识的灾难性遗忘 (Catastrophic Forgetting)。

接下来，我们通过图 6.8 中的例子来描述 RAG 的基本工作流程。用户输入一个问题“**2023 年的考拉数量有多少？**”，首先，该问题会传递给 RAG 框架的检索器模块，检索器从知识库中检索相关的知识文档，其中包含了与 2023 年的考拉数量相关的信息；接下来，这些信息通过 Prompt 的形式传递给大语言模型（大语言模型利用外部知识的形式是多样的，通过 Prompt 进行上下文学习是最常用的形

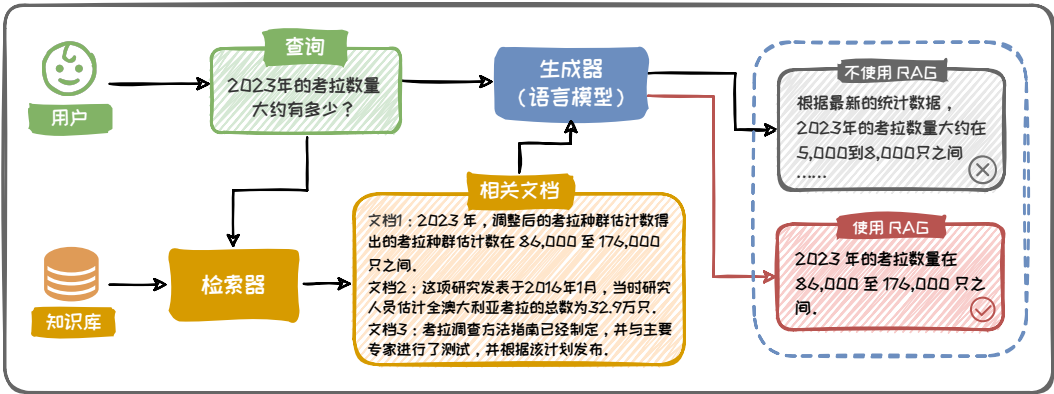


图 6.8: RAG 基本架构示意图。

式)，最终得出了正确的答案：“2023 年的考拉数量在 86,000 至 176,000 只之间。”然而，同样的问题，如果让大语言模型在不使用 RAG 的情况下直接回答，则无法得到正确的答案，这说明了 RAG 系统的有效性。

仅仅简单地对外部知识库、检索器、大语言模型等功能模块进行连接，无法最大化 RAG 的效用。本章将围绕以下三个问题，探讨如何优化设计 RAG 系统。

- **如何优化检索器与大语言模型的协作？** 根据是否对大语言模型进行微调，我们将现有的 RAG 系统分为（1）**黑盒增强架构**，不访问模型的内部参数，仅利用输出反馈进行优化；（2）**白盒增强架构**，允许对大语言模型进行微调。详细内容将在 6.2 节介绍。
- **如何优化检索过程？** 讨论如何提高检索的质量与效率，主要包括：（1）**知识库构建**，构建全面高质量的知识库并进行增强与优化；（2）**查询增强**，改进原始查询，使其更精确和易于匹配知识库信息；（3）**检索器**，介绍常见的检索器结构和搜索算法；（4）**检索效率增强**，介绍用于提升检索效率的常用相似度索引算法；（5）**重排优化**，通过文档重排筛选出更有效的信息。详细内容将在 6.3 节介绍。
- **如何优化增强过程？** 讨论如何高效利用检索信息，主要包括：（1）**何时增强**，

确定何时需要检索增强，以提升效率并避免干扰信息；**(2) 何处增强**，讨论生成过程中插入检索信息的常见位置；**(3) 多次增强**，针对复杂与模糊查询，讨论常见的多次增强方式；**(4) 降本增效**，介绍现有的知识压缩和缓存加速策略。详细内容将在 6.4 节介绍。

本节初步介绍了**我们为什么需要 RAG** 和 **RAG 是什么**这两个问题。接下来的章节将针对上面提出的三个问题，对具体技术细节详细讨论。

6.2 检索增强生成架构

检索增强生成 (RAG) 系统是一个集成了外部知识库、检索器、生成器等多个功能模块的软件系统。针对不同的业务场景和需求，可以设计不同的系统架构来组合、协调这些模块，以优化 RAG 的性能。其中，检索器和生成器的协作方式对 RAG 性能的影响最为显著。这是因为在不同的协作方式下，检索器检索到的信息质量会有所不同，生成器生成的内容质量也会随之变化。此外，检索器和生成器之间的协作方式对系统的效率有很大影响。高效的协作能够减少延迟，提高系统的响应速度。本节将从**如何优化检索器与大语言模型的协作**这一角度出发，对经典 RAG 架构进行梳理和介绍。

6.2.1 RAG 架构分类

针对不同的业务场景，RAG 中的生成器可以选用不同的大语言模型，如 GPT-4[1]、LLaMA[49] 等。考虑到大语言模型的开源/闭源、微调成本等问题，RAG 中的大语言模型可以是参数不可感知/调节的“黑盒”模型，也可以是参数可感知和微调的“白盒”模型。例如，如果选用 GPT-4，由于其闭源性，在 RAG 过程中只能将其视为“黑盒”，只能利用其输出结果，而无法感知/微调其模型参数。如果选择

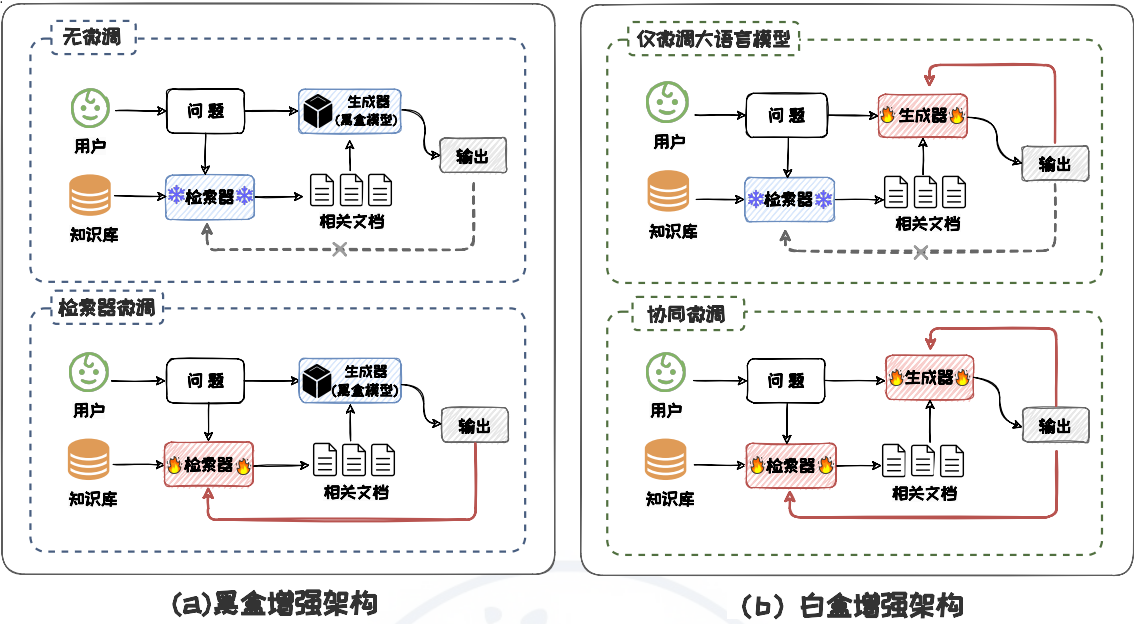


图 6.9: 检索增强架构分类图。其中含蓝色雪花的模块表示其参数被冻结、带红色火焰的部分表示微调时其参数被更新。

LLaMA 模型，在计算资源允许的情况下，在 RAG 过程中可将其视为“白盒”并对其进行微调。从是否对大语言模型进行微调的角度出发，本小节将 RAG 架构分类两大类：**黑盒增强架构**和**白盒增强架构**，如图6.9所示。

其中，黑盒增强架构可根据是否对检索器进行微调分为两类：**无微调**、**检索器微调**，如图6.9(a)所示。在无微调架构中，检索器和大语言模型都不进行任何微调，仅依靠它们在预训练阶段掌握的能力完成相应的检索和生成任务。在检索器微调的架构中，语言模型参数保持不变，而检索器根据语言模型的输出反馈进行参数的针对性调整。类似的，白盒增强架构也可根据是否对检索器进行微调分为两类：**仅微调大语言模型**、**检索器与大语言模型协同微调**（下文简称为协同微调），如图6.9(b)所示。在仅微调大语言模型的架构中，检索器作为一个预先训练好的组件其参数保持不变；语言模型则根据检索器提供的相关信息进行参数调整。在协同微调的架构中，检索器和大语言模型迭代交互、协同微调。

在 RAG 系统中，除了调整检索器和大语言模型，我们也可对其他功能模块（如知识库中的向量 [17, 45]）进行调整。调整其他功能模块与黑盒增强和白盒增强的分类是兼容的。本节接下来的部分将详细介绍黑盒增强架构和白盒增强架构，并探讨它们代表性方法。

6.2.2 黑盒增强架构

在某些情况下，由于无法获取大语言模型的结构和参数或者没有足够的算力对模型进行微调，例如只能通过 API 进行交互时，我们不得不将语言模型视为一个黑盒。此时，RAG 需要在黑盒增强架构的基础上构建。在黑盒增强架构中，我们仅可对检索器进行策略调整与优化。其可以分为无微调架构和检索器微调两种架构。接下来对两种架构类型分别展开介绍。

1. 无微调

无微调架构是所有 RAG 架构中形式最简单的。该架构中，检索器和语言模型经过分别独立的预训练后**参数不再更新，直接组合使用**。这种架构对计算资源需求较低，方便实现且易于部署，适合于对部署速度和灵活性有较高要求的场景。In-Context RALM[42] 是该框架下的代表性方法。其直接将检索器检索到的文档前置到输入问题前作为上下文，方法示意图如图6.10所示。In-Context RALM 包括检索和生成两个阶段。在检索阶段，输入的问题或部分句子作为查询从知识库中检索出相关文档。在生成阶段，这些检索到的文档被直接拼接到 Prompt 中的上下文部分，然后将 Prompt 输入给大语言模型。一个 RAG 任务可能涉及多次执行检索和生成。例如，在一个长文本生成任务中，每生成一定量的文本后，模型就可能会执行一次检索，以确保随着话题的发展，后续生成的内容能够持续保持与话题相关。

在执行检索操作时，需要仔细选择几个关键参数，如**检索步长**和**检索查询长度**。检索步长是指模型在生成文本时，每隔多少个词进行一次检索，这一参数的设

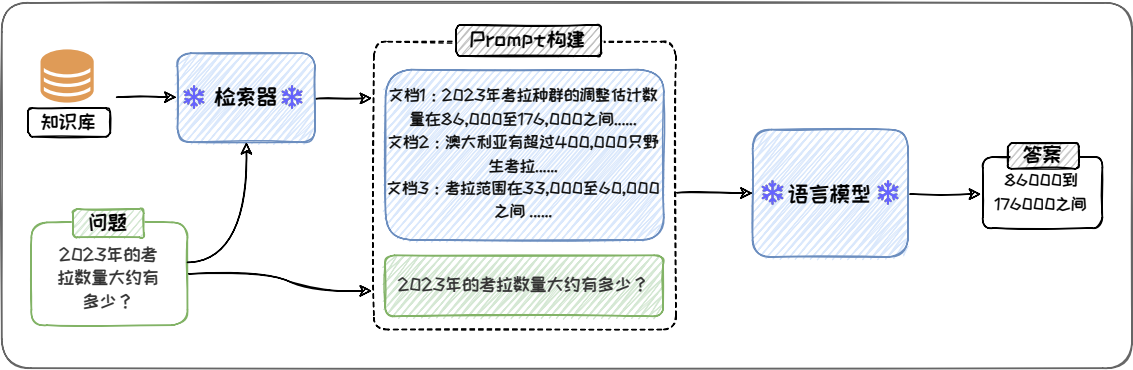


图 6.10: In-Context RALM 模型架构图。

定直接影响到模型的响应速度和信息的即时性。较短的检索步长能够提供更为及时的信息更新，但同时也可能增加计算的复杂性和资源消耗。因此，在实际应用中，需要在这两者之间找到一个合理的平衡点。检索查询长度指的是用于检索的文本片段的长度，通常被设置为语言模型输入中的最后几个词，以确保检索到的信息与当前的文本生成任务高度相关。

2. 检索器微调

虽然无微调架构在实现和部署上非常便捷，但它完全没有考虑检索器与语言模型之间潜在的协同效应，效果有待提升。为了进一步提升效果，可以采用检索器微调架构对检索器进行微调，以更好地适用于黑盒增强的环境。在检索器微调架构中，大语言模型的**参数保持不变**，仅用其输出指导检索器的微调。这种架构下的检索器能更好地适应大语言模型的需求，从而提高 RAG 的表现。

REPLUG LSR[45] 是检索器微调框架的代表性方法，其结构如图6.11所示。它使用大语言模型的困惑度分数作为监督信号来微调检索器，使其能更有效地检索出能够显著降低语言模型困惑度的文档。其微调检索器的过程中采用 KL 散度损失函数来训练检索器，目的是对齐检索到的文档的相关性分布与这些文档对语言模型性能提升的贡献分布。此过程涉及两个关键的概率分布，第一个是**检索器输出的文档分布**：检索器在接收到当前上下文后检索与之相关的文档，并形成一

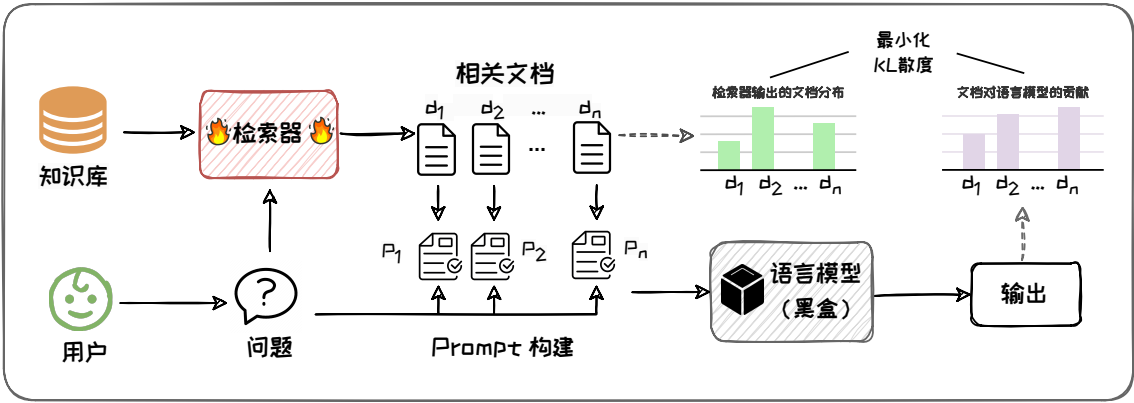


图 6.11: REPLUG LSR 模型架构图。

档概率分布。这一分布是基于检索器计算的上下文与文档之间的相似度，通过余弦相似度来衡量，并将这些相似度分数转化为概率值。第二个是**文档对语言模型的贡献分布**：语言模型为每个被检索到的文档和原始上下文来生成预测，最终所有输出结果形成一个概率分布。在这个分布中，如果某个文档对语言模型生成准确预测特别关键，它会被赋予更高的概率权重。在微调过程中，REPLUG LSR 将语言模型视为黑盒处理，仅通过模型的输出来指导检索器的训练，避免了对语言模型内部结构的访问和修改。此外，在微调过程中，REPLUG LSR 还采用了一种异步索引更新策略，即不会在每次训练步骤后立即更新知识库的向量编码，而是在一定的训练步骤之后才进行更新。这种策略降低了索引更新的频率，减少了计算成本，使模型能够在连续训练过程中更好地适应新数据。此外，检索器微调框架中还可以引入代理模型来指引检索器微调。例如，AAR[60]方法通入引入额外的小型语言模型，使用它的交叉注意力得分标注偏好文档，以此来微调检索器，使其能够在不微调目标语言模型的情况下增强其在不同任务上的表现。

检索器微调的方式允许即使是闭源大模型如 ChatGPT，也能通过优化外部检索器来提升性能。REPLUG LSR 和 AAR 通过该方式实现了在保持大模型完整性的同时，通过外部调整来增强模型的能力，这在传统的 RAG 中是不常见的。

6.2.3 白盒增强架构

通常，大语言模型和检索器是独立预训练的，二者可能存在匹配欠佳的情况。白盒增强架构通过微调大语言模型来配合检索器，以提升 RAG 的效果。其可根据是否对检索器进行微调分为两类：**仅语言模型微调**、**检索器和语言模型协同微调**。

1. 仅微调语言模型

仅微调语言模型指的是检索器作为一个预先训练好的组件其参数保持不变，大语言模型根据检索器提供的上下文信息，对**自身参数进行微调**。RETRO[5] 是仅微调语言模型的代表性方法之一。该方法通过修改语言模型的结构，使其在微调过程中能够将从知识库中检索到的文本直接融入到语言模型中间状态中，从而实现外部知识对大语言模型的增强。此外，SELF-RAG[3] 通过在微调语言模型时引入反思标记，使语言模型在生成过程中动态决定是否需要检索外部文本，并对生成结果进行自我批判和优化。这些方法不仅提高了生成内容的质量和事实准确性，还增强了模型的知识整合与应用能力。

以 RETRO 为例，其结构如图6.12所示。RETRO 首先将知识库中的文本进行切块，然后用 BERT 对每个文本块生成嵌入向量。在微调模型时的自回归过程中，每当模型生成一段文本块后，就去知识库中检索出与之最相似的嵌入向量。然后，这些嵌入向量和模型注意力层的输出一起被送入一个外部的 Transformer 编码器进行编码。得到的编码向量直接输入给模型的块交叉编码器的键（key）和值（value），以捕捉外部知识的关键信息。通过交叉编码，模型能够结合检索到的相关信息来生成新的文本块。

通过上述方式微调后的 RETRO 模型能够充分整合检索到的信息，生成连贯且富含信息的文本。面对用户查询时，模型能展现出优秀的理解能力和知识整合能力，大幅提升了生成的质量和准确性，尤其在处理复杂任务时，其表现更为突出。

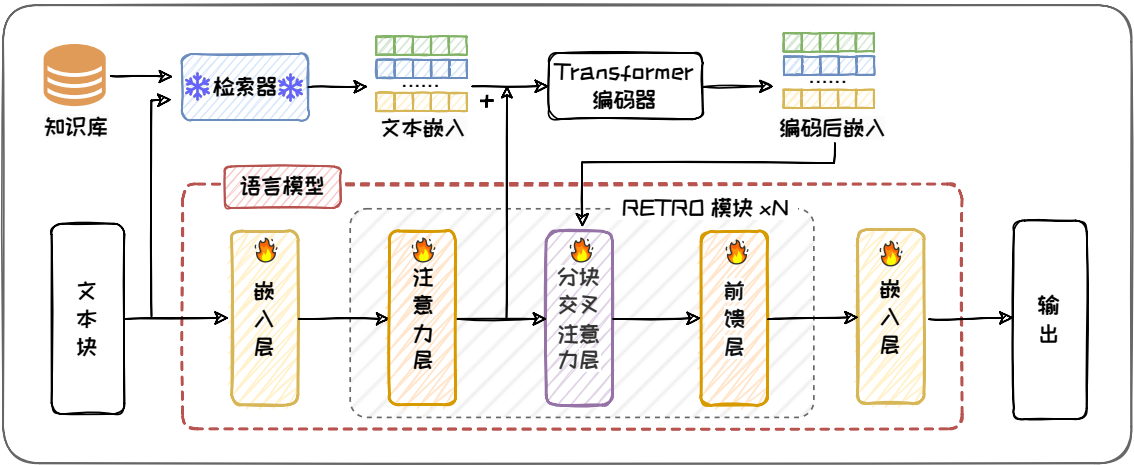


图 6.12: RETRO 模型架构图。

2. 检索器和语言模型协同微调

在仅微调语言模型的架构下，检索器作为固定组件，微调过程中其参数保持不变。这导致检索器无法根据语言模型的需求进行适应性调整，从而限制了检索器与语言模型之间的相互协同。在检索器和语言模型协同微调的架构中，检索器和语言模型的参数更新同步进行。这种微调的方式使得检索器能够在检索的同时学习如何更有效地支持语言模型的需求，而语言模型则可以更好地适应并利用检索到的信息，以进一步提升 RAG 的性能。

Atlas[17] 是该架构的代表性工作，其架构如图 6.13 所示。与 REPLUG LSR 类似，其在预训练和微调阶段使用 KL 散度损失函数来联合训练检索器和语言模型，以确保检索器输出的文档相关性分布与文档对语言模型的贡献分布相一致。不同之处在于，Atlas 在预训练和微调过程中，检索器和语言模型参数同步被更新，检索器学习向语言模型提供最相关的文档，而语言模型则学习如何利用这些文档来改善其对查询的响应。为了确保检索结果与模型最新状态保持同步，Atlas 同样需要定期更新语料库文档的向量编码，从而维持检索的准确性。

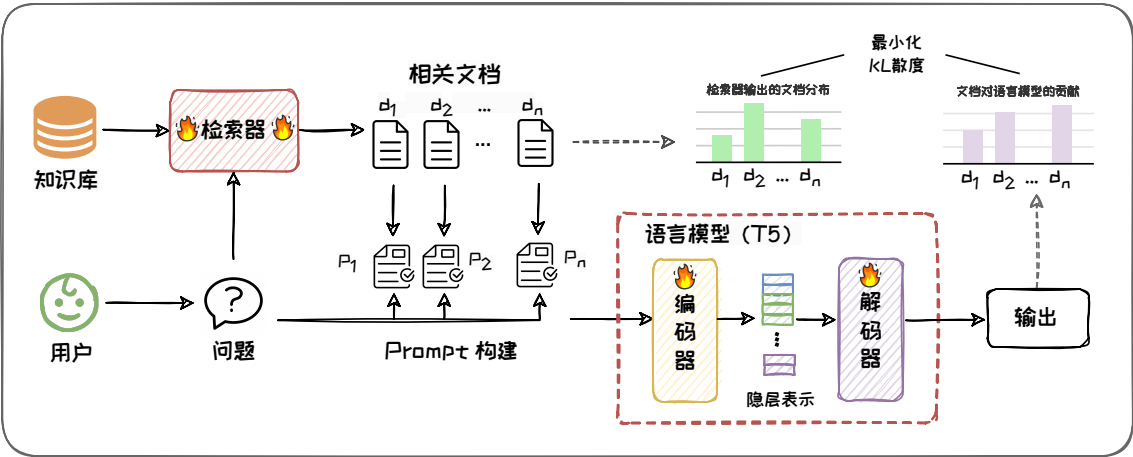


图 6.13: Atlas 模型架构图。

6.2.4 对比与分析

本节主要介绍了 RAG 的黑盒增强架构和白盒增强架构及其经典方法。接下来，我们对这两种架构进行总结和对比。

黑盒增强架构是在闭源模型的背景下提出的，它限制了对模型内部参数的直接调整。在这种架构下，我们介绍了**无微调**和**检索器微调**两种策略。**无微调**简单实用，它直接利用预训练的语言模型和检索器，不进行任何更新，适合快速部署。然而，这种方法的缺点在于无法对语言模型进行优化以适应新的任务需求。相比之下，**检索器微调**通过调整检索器来适应语言模型输出，提供了在无法修改语言模型的情况下提升性能的可能性。这种方法的效果在很大程度上取决于调整后的检索器的准确性。

白盒增强架构则利用开源模型的优势，允许调整语言模型结构和参数，可以更好的协调减速器和大语言模型。在这种架构中，我们介绍了两种微调形式：**仅微调语言模型**和**检索器和语言模型协同微调**。**仅微调语言模型**专注于优化语言模型，根据检索到的信息仅调整语言模型结构和参数，以提升特定任务上的性能。**检索器和语言模型协同微调**是一种更为动态的策略，它通过同步更新检索器和语言模

型，使得两者能够在训练过程中相互适应，从而提高整体系统的性能。尽管白盒增强架构可以有效改善 RAG 的性能，但也存在明显缺点。这种架构通常需要大量计算资源和时间来训练，特别是协同微调策略，需要大量的运算资源来实现语言模型和检索器的同步更新。

6.3 知识检索

在 RAG 中，检索的效果（召回率、精度、多样性等）会直接影响大语言模型的生成质量。以“树袋熊一般在哪里生活？”这个问题为例。如果检索器返回的外部知识是关于“树袋熊”名称相近的动物“袋熊”的相关知识，那么这些不正确的外部知识可能引导大语言模型生成错误答案。此外，检索的时间也是 RAG 总耗时的关键部分，因此检索的效率将影响用户的使用体验。优化检索过程，提升检索的效果和效率，对改善 RAG 的性能具有重要意义。针对优化检索过程，本节系统的对知识库构建、查询增强、检索器、检索结果重排序等关键技术进行梳理和介绍。

6.3.1 知识库构建

知识库构成了 RAG 系统的根基。正如古语所言：“巧妇难为无米之炊”，只有构建了全面、优质、高效的知识库，检索才能有的放矢，检索效果才能有保障。在 RAG 框架中，知识库构建主要涉及**数据采集及预处理**与**知识库增强**两个步骤。本小节将对这两个步骤分别展开介绍。

1. 数据采集及预处理

数据采集与预处理为构建知识库提供“原材料”。在构建文本型知识库的**数据采集**过程中，来自不同渠道的数据被整合、转换为统一的文档对象。这些文档对象不仅包含原始的文本信息，还携带有关文档的元信息（Metadata）。元信息可以用

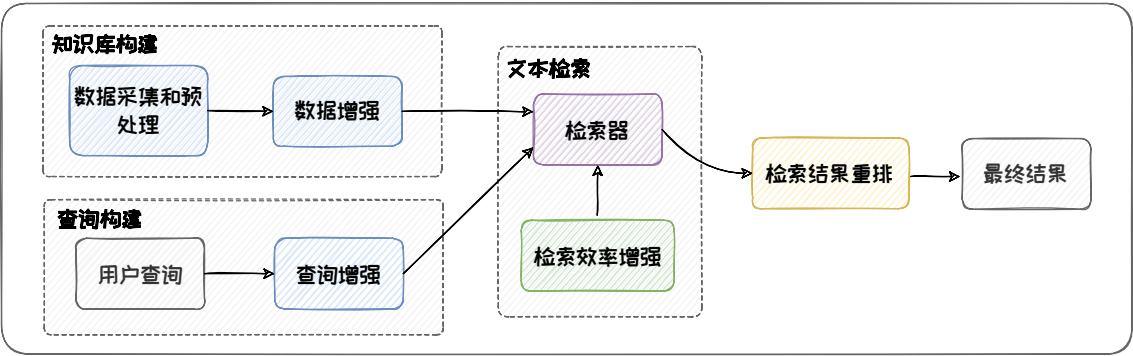


图 6.14: 知识检索流程图。

于后续的检索和过滤。以维基百科语料库的构建为例，数据采集主要通过提取维基百科网站页面内容来实现。这些内容不仅包括正文描述的内容，还包括一系列的元信息，例如文章标题，分类信息，时间信息，关键词等。

在采集到相应的数据后，还需通过**数据预处理**来提升数据质量和可用性。在构建文本型知识库时，数据预处理主要包括数据清洗和文本分块两个过程。**数据清洗**旨在清除文本中的干扰元素，如特殊字符、异常编码和无用的 HTML 标签，以及删除重复或高度相似的冗余文档，从而提高数据的清晰度和可用性。**文本分块**是将长文本分割成较小文本块的过程，例如把一篇长文章分为多个短段落。对长文本进行分块有两个好处：一是为了适应检索模型的上下文窗口长度限制，避免超出其处理能力；二是通过分块可以减少长文本中的不相关内容，降低噪音，从而提高检索的效率和准确性。

文本分块的效果直接影响后续检索结果的质量 [14]。如果分块处理不当，可能会破坏内容的连贯性。因此，制定合适的分块策略至关重要，包括确定切分方法（如按句子或段落切分）、设定块大小，以及是否允许块之间有重叠。文本分块的具体实施流程通常开始于将长文本拆解为较小的语义单元，如句子或段落。随后，这些单元被逐步组合成更大的块，直到达到预设的块大小，构建出独立的文本片段。为了保持语义连贯性，通常还会在相邻的文本片段之间设置一定的重叠区域。

2. 知识库增强

知识库增强是通过改进和丰富知识库的内容和结构，以提升其质量和实用性。这一过程通常涉及**查询生成与标题生成** [56] 等多个步骤，以此为文档建立语义“锚点”，方便检索时准确定位到相应文本。

查询生成指的是利用大语言模型**生成与文档内容紧密相关的伪查询**。这些伪查询从查询的角度来表达文档的语义，可以作为相关文档的“键”，供检索时与用户查询进行匹配。通过这种方式，可以增强文档与用户查询的匹配度。例如，对于一篇介绍考拉和树袋熊关系的文档，生成的查询“考拉和树袋熊之间的关系是什么？”不仅准确反映了文档的主题，还能有效引导检索器更精确的检索到与用户提问相关的信息。

标题生成指的是利用大语言模型**为没有标题的文档生成合适的标题**。这些生成的标题提供了文档的关键词和上下文信息，能来用来帮助快速理解文档内容，并在检索时更准确地定位到与用户提问相关的信息。对于那些原始文档中缺乏标题的情况，通过语言模型生成标题显得尤为重要。

6.3.2 查询增强

知识库涵盖的知识表达形式是有限的，但用户的提问方式却是千人千面的。用户遣词造句的方式以及描述问题的角度可能会与知识库中的存储的文本间存在差异，这可能导致用户查询和知识库之间不能很好匹配，从而降低检索效果。为了解决此问题，我们可以对用户查询的语义和内容进行扩展，即查询增强，以更好的匹配知识库中的文本。本小节将从**查询语义增强**和**查询内容增强**两个角度出发，对查询增强技术进行简要介绍。

1. 查询语义增强

查询语义增强旨在通过**同义改写**和**多视角分解**等方法来扩展、丰富用户查询

的语义，以提高检索的准确性和全面性。接下来分别对同义改写和多视角分解进行简要介绍。

(1) 同义改写

同义改写通过将原始查询改写成相同语义下不同的表达方式，来解决用户查询单一的表达形式可能无法全面覆盖到知识库中多样化表达的知识。改写工作可以调用大语言模型完成。比如，对于这样一个原始查询：“考拉的饮食习惯是什么？”，可以改写成下面几种同义表达：1、“考拉主要吃什么？”；2、“考拉的食物有哪些？”；3、“考拉的饮食结构是怎样的？”。每个改写后的查询都可独立用于检索相关文档，随后从这些不同查询中检索到的文档集合进行合并和去重处理，从而形成一个更大的相关文档集合。

(2) 多视角分解

多视角分解采用分而治之的方法来处理复杂查询，将复杂查询分解为来自不同视角的子查询，以检索到查询相关的不同角度的信息。例如，对于这样一个问题：“考拉面临哪些威胁？”，可以从多个视角分解为：1、“考拉的栖息地丧失对其有何影响？”；2、“气候变化如何影响考拉的生存？”；3、“人类活动对考拉种群有哪些威胁？”；4、“自然灾害对考拉的影响有哪些？”等子问题。每个子问题能检索到不同的相关文档，这些文档分别提供来自不同视角的信息。通过综合这些信息，语言模型能够生成一个更加全面和深入的最终答案。

2. 查询内容增强

查询内容增强旨在通过生成与原始查询相关的背景信息和上下文，从而丰富查询内容，提高检索的准确性和全面性 [58]。与传统的仅依赖于检索的方式相比，查询内容增强方法通过引入大语言模型生成的辅助文档，为原始查询提供更多维度的信息支持。

生成背景文档是一种查询内容增强的方法。它指的是在原始查询的基础上，利

用大语言模型生成与查询内容相关的背景文档。例如，对于用户查询“如何保护考拉的栖息地?”，可以生成以下背景文档：

考拉是原产于澳大利亚的树栖有袋类动物，主要分布在东部和东南部沿海的桉树林中。这些地区提供了考拉主要食物来源——桉树叶。考拉的栖息地包括开阔的森林和木林地，这里桉树丰富，不仅提供食物，还提供栖息和保护。考拉高度依赖特定种类的桉树，它们的分布与这些树木的可用性密切相关。

这些生成的背景文档可以作为原始查询的补充信息，提供更多的上下文内容，从而提高检索结果的相关性和丰富性。

6.3.3 检索器

给定知识库和用户查询，检索器旨在找到知识库中与用户查询相关的知识文本。检索器可分为**判别式检索器**和**生成式检索器**两类。本小节将对这两类检索器分别展开介绍。

1. 判别式检索器

判别式检索器通过判别模型对查询和文档是否相关进行打分。判别式检索器通常分为两大类：**稀疏检索器**和**稠密检索器**。稀疏检索器利用离散的、基于词频的文档编码向量进行检索，而稠密检索器则利用神经网络生成的连续的、稠密向量对文档进行检索。下面将详细的介绍这两种检索器以及代表性方法。

(1) 稀疏检索器

稀疏检索器（Sparse Retriever）是指使用**稀疏表示方法**来匹配文本的模型。这类检索器通过统计文档中特定词项出现的统计特征来对文档进行编码，然后基于此编码计算查询与知识库中的文档的相似度来进行检索。典型的稀疏检索技术包

括 TF-IDF[2] 和 BM25[43] 等，它们通过分析词项的分布和频率来评估文档与查询的相关性。TF-IDF 基于词频 (TF) 和逆文档频率 (IDF) 来衡量词语在文档或语料库中的重要性，然后用此重要性对文本进行编码。词频 (TF) 表示词语在文档中的出现频率，计算公式为：

$$\text{tf}_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}, \quad (6.1)$$

其中， $n_{i,j}$ 是词语 t_i 在文档 d_j 中的出现次数， $\sum_k n_{k,j}$ 是文档 d_j 中所有词语的出现次数之和，用于进行标准化以避免偏向长文档。逆文档频率 (IDF) 衡量词语的普遍性，计算公式为：

$$\text{idf}_i = \log \frac{|D|}{|\{j : t_i \in d_j\}|}, \quad (6.2)$$

其中， $|D|$ 是总文档数， $|\{j : t_i \in d_j\}|$ 是包含词语 t_i 的文档数。最终，TF-IDF 值为：

$$\text{tfidf}_{i,j} = \text{tf}_{i,j} \times \text{idf}_i. \quad (6.3)$$

TF-IDF 通过高词频和低文档频率产生高权重，倾向于过滤常见词语，保留重要词语。

BM25 是一种改进的文本检索算法，它在 TF-IDF 基础上通过文档长度归一化和词项饱和度调整，更精确地评估词项重要性，优化了词频和逆文档频率的计算，并考虑了文档长度对评分的影响。虽然不涉及词项上下文，但是 BM25 在处理大规模数据时表现优异，广泛应用于搜索引擎和信息检索系统。

(2) 稠密检索器

稠密检索器一般利用 **预训练语言模型** 对文本生成 **低维、密集** 的向量表示，通过计算向量间的相似度进行检索。按照所使用的模型结构的不同，稠密检索器大致可以分为两类：**交叉编码类** (Cross-Encoder)、**双编码器类** (Bi-Encoder)。二者结构分别如图 6.15 (a) 和 6.15 (b) 所示。

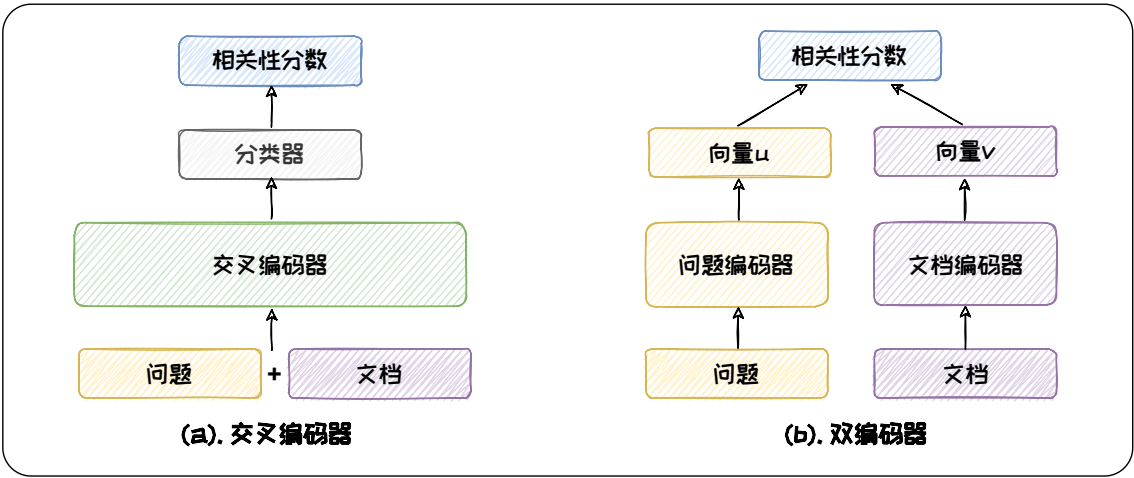


图 6.15: 不同稠密检索器对比图。

交叉编码类

交叉编码类“端到端”的给出查询和文档的相似度。这类模型将查询和文档拼接在一起，随后利用预训练语言模型作为编码器（例如 BERT）生成一个向量表示。接着，通过一个分类器处理这个向量，最终输出一个介于 0 和 1 之间的数值，表示输入的查询和文档之间的相似程度。其优点在于模型结构简单，能够实现查询和文档之间的深度交互，例如，在工作 [12, 44] 中，研究者们使用了交叉编码器来提升检索性能。然而，由于交叉编码类模型需要进行高复杂度的交叉注意力操作，计算量大，因此不适合在大规模检索阶段使用。这种模型更适用于对少量候选文档进行更精确排序的阶段，可以显著提升检索结果的相关性。

双编码器类

与交叉编码类模型不同，双编码类模型采用了一种“两步走”的策略。第一步，查询和文档首先各自通过独立的编码器生成各自的向量表示；第二步，对这两个向量之间的相似度进行计算，以评估它们的相关性。这种方法的优势在于，它允许预先离线计算并存储所有文档的向量表示，在线检索时则可直接进行向量匹配。因此，双编码器非常适合在工业环境中部署，具有极高的匹配效率。然而，在这种分离的处理方式中，查询与文档在提取特征向量时缺乏交互。这可能会对匹配的精度

确度产生影响。DPR (Dense Passage Retriever) [23] 是稠密检索器的一个代表工作。其使用两个独立的 BERT 编码器，分别将查询和文档映射到低维特征向量，然后通过向量点积衡量相似度。为了缓解查询与文档缺乏交互的问题，DPR 通过对比学习优化编码器，最大化查询与相关段落相似度的同时最小化与负面段落相似度。

为了缓解查询与文档在提取特征向量时缺乏交互的问题，可以在双编码器的基础上引入查询与文档的交互，以进一步提升双编码器的效果。ColBERT[24] 是其中的代表性方法。其以查询和文档间的 Token 级的相似度为度量，然后通过对比学习对双编码器进行微调，以使双编码器编码的特征向量可以兼顾查询和文档。此外，在 RAG 中，我们有时会对查询加入大段上下文进行增强，如第 3.3.2 节所介绍的情形。此时，查询的长度急剧增长，传统的检索方法可能难以有效的处理这些长查询。为解决此问题，可以采用 Poly-encoder[16]。其模型架构沿用双编码器的形式，但是它使用 m 个向量来捕获长查询的多个特征，而不是像普通双编码器那样只用一个向量来表示整个查询。并且，这 m 个向量随后与文档的向量通过注意力机制进行交互，其中的注意力模块采用查询和文档间的对比学习进行训练。

2. 生成式检索器

生成式检索器通过生成模型对输入查询直接生成相关文档的标识符 [29]。与判别式检索器不断地从知识库中去匹配相关文档不同，生成式检索器直接将知识库中的文档信息记忆在模型参数中。然后，在接收到查询请求时，能够直接生成相关文档的标识符（即 DocID），以完成检索 [48]。生成式检索器通常采用基于 Encoder-Decoder 架构的生成模型，如 T5[41]、BART[27] 等。生成式检索器的训练过程通常分为两个阶段 [29]。在第一阶段，模型通过序列到序列的学习方法，学习如何将查询映射到相关的文档标识符。这一阶段主要通过最大似然估计 (MLE) 来优化模型，确保生成的文档标识符尽可能准确。在第二阶段，通过数据增强和排名优化进一步提高检索效率和准确性。数据增强主要通过生成伪查询 [51] 或使用文

档片段 [61] 作为查询输入，以增加训练数据的多样性和覆盖面。排名优化则涉及使用特定的损失函数，如对比损失或排名损失，来调整模型生成文档标识符的顺序和相关性，从而更好地匹配查询的需求。

在生成式检索器中，DocID 的设计至关重要。其需要在语义信息的丰富性与标识符的简洁性之间取得平衡。常用的 DocID 形式分为两类：基于数字的 DocID 和基于词的 DocID。基于数字的 DocID 方法使用唯一的数字值或整数字符串来表示文档，虽然构建简单，但在处理大量文档时可能导致标识符数量激增，增加计算和存储负担。相比之下，基于词的 DocID 方法直接从文档的标题、URL 或 N-gram 中提取表示 [9]，能更自然地传达文档的语义信息。通常，标题是最佳选择，因为它提供了文档的宏观概述。但在缺乏高质量标题时，URL 或 N-gram 也可作为有效的替代方案。

尽管生成式检索器在性能上取得了一定的进步，但与稠密检索器相比，其效果仍稍逊一筹。此外，生成式检索器还面临着一系列挑战，包括如何突破模型输入长度的限制、如何有效处理大规模文档以及动态新增文档的表示学习等，这些都是亟待解决的问题。

6.3.4 检索效率增强

知识库中通常包含海量的文本，对知识库中文本进行逐一检索缓慢而低效。为提升检索效率，可以引入向量数据库来实现检索中的高效向量存储和查询 [39]。向量数据库的核心是设计高效的相似度索引算法。本节将简要介绍常用的相似度索引算法，以及用于构建向量数据库的常见软件库。

1. 相似度索引算法

在向量检索中，常用的索引技术主要分成三大类：基于空间划分的方法、基于量化方法和基于图的方法。

基于空间划分的方法将搜索空间划分为多个区域来实现索引，主要包括基于树的索引方法和基于哈希的方法两类。其中，基于树的索引方法通过一系列规则递归地划分空间，形成一种树状结构，每个叶节点代表一个较小区域，区域内的数据点彼此接近。在查询时，算法从树的根节点出发，逐步深入到合适的叶节点，最后在叶节点内部进行数据点的相似度比较，以找到最近的向量。常见的基于树的索引包括 KD 树 [4] 和 Ball 树 [13] 等。而基于哈希的方法（如局部敏感哈希（LSH）[11]）通过哈希函数将向量映射到哈希表的不同桶中，使得相似向量通常位于同一桶内。

基于图的方法通过构建一个邻近图，将向量检索转化为图的遍历问题。这类方法在索引构建阶段，将数据集中的每个向量表示为图中的一个节点，并根据向量间的距离或相似性建立边的连接。不同的图索引结构主要体现在其独特的赋边策略上。索引构建的核心思想源于小世界网络模型，旨在创建一个结构，使得从任意入口点出发，能在较少步数内到达查询点的最近邻。这种结构允许在搜索时使用贪婪算法，逐步逼近目标。然而，图索引设计面临稀疏性和稠密性的权衡：较稀疏的图结构每步计算代价低，而较稠密的图则可能缩短搜索路径。基于图的代表性方法有 NSW[32]、IPNSW[37] 和 HNSW[31] 等。

基于乘积量化的方法通过将高维向量空间划分为多个子空间，并在每个子空间中进行聚类得到码本和码字，以此作为构建索引的基础 [18]。这类方法主要包括训练和查询两个阶段。在训练阶段，该方法学习如何将高维向量最优地量化为码字 ID 序列。通常这个过程涉及将原始空间划分为多个子空间，在每个子空间内进行聚类，并通过聚类中心得到码字和码本。每个子空间内的聚类中心点即为码字，所有码字的集合构成码本。每个训练样本的每个子向量可以都用相应子空间的码字来近似，这样就实现了码字 ID 序列来表示训练样本，达到了数据量化的目的。在查询阶段，系统同样将查询向量划分为子向量，并在每个子空间中找到最近的码

字，得到码字 ID 序列。随后，系统计算查询向量的每个子向量到所有对应子空间的码字的距离，形成距离表。最后，系统利用这个距离表和数据库中每个向量的码字 ID 序列，快速查找并累加各个子向量的对应距离，得到查询向量与数据库向量之间的近似距离。通过对这些距离进行排序，系统最终得到最近邻结果。该方法在减少内存占用和加快距离计算速度方面表现出色，但量化过程中会不可避免地会引入一些误差。在某些需要精度更高的应用场景中，我们可以在 PQ 的基础上进一步进行精确排序，以得到精确的最近邻结果。此外，还有一些乘积量化的优化算法以及和其他索引相结合的算法，如 OPQ[15]、IVFPQ[19] 等。

2. 常见软件库介绍

在前文中，我们已经介绍了向量数据库的核心技术——相似度索引算法。这些算法是实现高效向量检索的关键。接下来，我们将介绍几种常用于构建和管理向量数据库的软件库，它们支持上述的相似性索引算法，是实现高效向量检索的重要工具。

Faiss⁴，由 Meta AI Research 开发，是一个专门优化密集向量相似性搜索和聚类的库。Faiss 提供了多种索引算法选择，这些算法涵盖了基于空间划分、基于量化以及基于图的方法等。这些算法不仅能在 CPU 上运行，部分算法还支持 GPU 加速，从而满足不同应用场景下的性能需求。然而，Faiss 本身并不是一个完整的数据库系统，而是一个功能强大的工具库。它专注于提供高效的索引和搜索功能，但在数据存储、管理、分布式支持和安全性措施等方面，Faiss 的功能相对有限。相比之下，向量数据库是一种更全面的解决方案，它不仅包括相似度索引算法，还整合了数据存储、管理、分布式支持和安全性措施等多方面的功能。截至目前，市场上已有多款成熟的向量数据库，如表6.1所示，它们适用于各种更复杂的 RAG 应用场景。

⁴<https://github.com/facebookresearch/faiss>

表 6.1: 常见的向量数据库。

向量数据库	URL	GitHub Star
milvus	https://github.com/milvus-io/milvus	28.4K
typesense	https://github.com/typesense/typesense	19.0K
qdrant	https://github.com/qdrant/qdrant	18.9K
chroma	https://github.com/chroma-core/chroma	13.7K
weaviate	https://github.com/weaviate/weaviate	10.4K
pinecone	https://www.pinecone.io/	×

6.3.5 检索结果重排

检索器可能检索到与查询相关性不高的文档。这些文档如果直接输入给大语言模型，可能会引发生成质量的下降。为此，在将其输入给大语言模型之前，我们还需要对其进行进一步的精选。精选的主要途径是对检索到的文档进行重新排序，简称重排，然后从中选择出排序靠前的文档。重排方法主要分为两类：基于交叉编码的方法和基于上下文学习的方法。

1. 基于交叉编码的重排方法

基于交叉编码的重排方法利用**交叉编码器**（Cross-Encoders）来评估文档与查询之间的语义相关性。关于交叉编码的介绍见第6.3.3节。MiniLM-L⁵是应用最为广泛的基于交叉编码的重排开源模型之一。该模型通过减少层数和隐层单元数来降低参数数量，同时采用知识蒸馏技术从大型、高性能的语言模型中继承学习，以此来提高模型性能。此外，还有其他一些在线重排模型可通过 API 直接访问，例如 Cohere⁶。对于希望探索其他高性能的重排器的读者，可以参考 MTEB⁷ 排行榜，该榜单汇集了很多性能优秀的重排模型。

2. 基于上下文学习的重排方法

基于上下文学习的方法是指通过设计精巧的 Prompt，使用大语言模型来执行

⁵<https://huggingface.co/cross-encoder/ms-marco-MiniLM-L-6-v2>

⁶<https://cohere.com/rerank>

⁷<https://huggingface.co/spaces/mteb/leaderboard>

RankGPT Prompt 模板

这是RankGPT，一款智能助手，专门用于根据查询的相关性对段落进行排序。

以下是{{num}}段文字，每段都有一个数字标识符[]。我将根据查询内容对它们进行排序：{{query}}

[1] {{passage_1}}
[2] {{passage_2}}
(更多段落) ...

查询是：{{query}}

我将根据查询对上述{{num}}段文字进行排序。这些段落将按照相关性降序列出，使用标识符表示，最相关的段落将列在最前面，输出格式应该是[] > [] > 等，例如，[1] > [2] > 等。

对{{num}}段文字的排序结果（仅限标识符）是：

图 6.16: RankGPT Prompt 模板图。

重排任务。这种方法可以利用大语言模型优良的深层语义理解能力，从而取得了良好的表现。RankGPT[47] 是基于上下文学习的重排方法中的代表性方法。其使用的 Prompt 模板如图6.16所示。在重排任务中，输入文档长度有时会超过上下文窗口长度的限制。为了解决该问题，RankGPT 采用了滑动窗口技术来优化排序过程。该技术将所有待排序的文档分割成多个连续的小部分，每个部分作为一个窗口。整个排序过程从文档集的末尾开始：首先，对最后一个窗口内的文档进行排序，并将排序后的结果替换原始顺序。然后，窗口按照预设的步长向前移动，重复排序和替换的过程。这个过程将持续进行，直到所有文档都被处理和排序完毕。通过这种分步处理的方法，RankGPT 能够有效地对整个文档集合进行排序，而不受限于单一窗口所能处理的文档数量。

6.4 生成增强

检索器得到相关信息后，将其传递给大语言模型以期增强模型的生成能力。利用这些信息进行生成增强是一个复杂的过程，不同的方式会显著影响 RAG 的性能。本节将从如何优化增强过程这一角度出发，围绕四个方面展开讨论：(1) 何时增强，确定何时需要检索增强，以确保非必要不增强；(2) 何处增强，确定在模型中

的何处融入检索到的外部知识，以最大化检索的效用；(3) **多次增强**，如何对复杂查询与模糊查询进行多次迭代增强，以提升 RAG 在困难问题上的效果；(4) **降本增效**，如何进行知识压缩与缓存加速，以降低增强过程的计算成本。

6.4.1 何时增强

大语言模型在训练过程中掌握了大量知识，这些知识被称为**内部知识 (Self-Knowledge)**。对于内部知识可以解决的问题，我们可以不对该问题进行增强。不对是否需要增强进行判断而盲目增强，不仅不会改善生成性能，还可能“画蛇添足”引起**生成效率**和**生成质量**上的双下降。对**生成效率**而言，增强文本的引入会增加输入 Token 的数量，增加大语言模型的推理计算成本。另外，检索过程也涉及大量的计算资源。对**生成质量**而言，因为检索到的外部知识有时可能存在噪音，将其输入给大语言模型不仅不会改善大语言模型的生成质量，反而可能会生成错误内容。如图 6.17 所示，对于“树袋熊一般在哪里生活？”这个问题，大语言模型可以直接给出正确答案。但是，当我们为它提供一段与树袋熊名称非常相似的动物袋熊的外部知识，如图 6.18 所示，大语言模型给出了错误答案，因为大语言模型将知识文本中关于袋熊的信息错误地理解为树袋熊的相关信息。综上，判断大语言模型何时需要检索增强，做到非必要不增强，可以有效的降低计算成本并避免错误增强。

判断是否需要增强的核心在于**判断大语言模型是否具有内部知识**。如果我们判断大模型对一个问题具备内部知识，那么我们就可以避免检索增强的过程，不仅降低了计算成本，而且还可以避免错误增强。判断模型是否具有内部知识的方法可以分为两类：(1) 外部观测法，通过 Prompt 直接询问模型是否具备内部知识，或应用统计方法对是否具备内部知识进行估计，这种方法无需感知模型参数；(2) 内部观测法，通过检测模型内部神经元的状态信息来判断模型是否存在内部知识，这种方法需要对模型参数进行侵入式的探测。

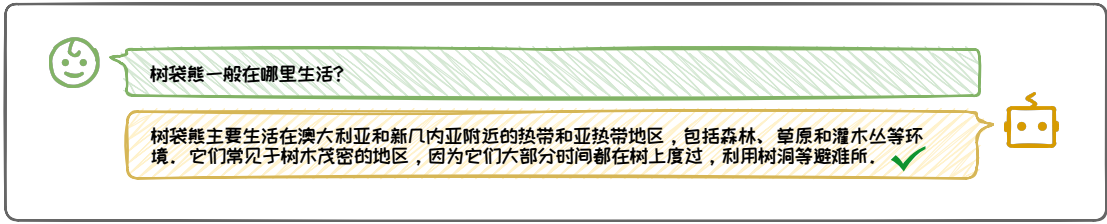


图 6.17: 模型已知问题示例。

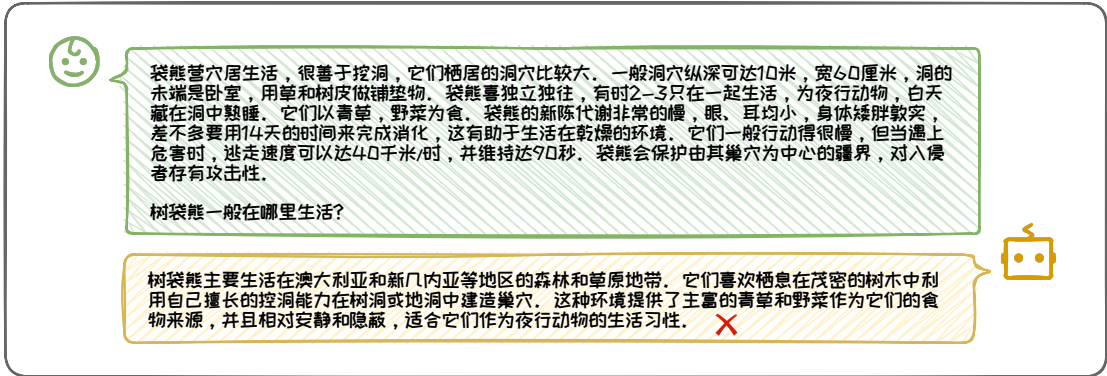


图 6.18: 知识文档损害性能示例。

1. 外部观测法

外部观测法旨在不侵入模型内部参数的情况下，通过直接对大语言模型进行询问或者观测调查其训练数据来推断其是否具备内部知识。这种方法可以类比为人类面试的过程。面试官在评估应聘者的知识和能力时，通常会询问并观察其反应、浏览其过往教育经历等方式，以判断应聘者是否具备足够的专业知识。对于大语言模型，我们可以通过两种问询的方式来判断大语言模型是否具备相应的内部知识：（1）Prompt 直接询问大语言模型是否含有相应的内部知识；（2）反复询问大语言模型同一个问题观察模型多次回答的一致性。此外，我们也可以通过翻看大语言模型的“教育经历”，即训练数据来判断其是否具备内部知识。但是，许多大语言模型的训练数据并未公开，无法直接观测它们的训练数据。在这种情况下，可以通过设计伪训练数据统计量来拟合真实训练数据的分布，从而间接评估模型对特定知识的学习情况。接下来将对这三种方式进行详细介绍。

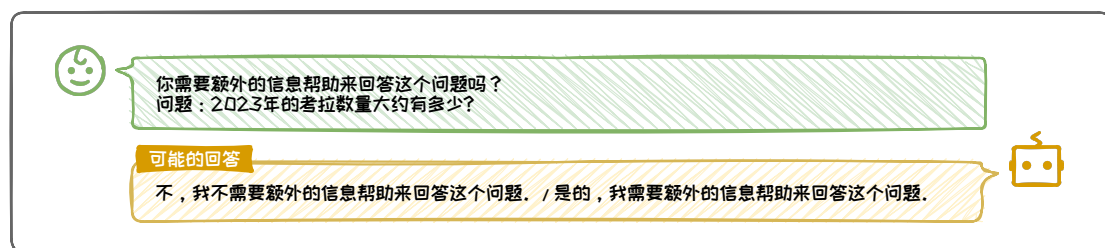


图 6.19: 直接询问的 Prompt 示例。

(1) 询问

我们可以通过直接询问和多次询问两种询问方法来判断大语言模型是否具备内部知识。在直接询问时，我们可以编写 Prompt 直接询问大语言模型是否需要外部知识，如图 6.19 所示。然而，大语言模型很难做到“知之为知之”，其存在“过度自信”的问题：对于无法回答的问题，大语言模型经常认为自己不需要外部知识的帮助，因此这种判断方式的准确率较低。此外，采用多次询问时，我们可以通过让大语言模型重复多次地回答同一个问题，然后根据其回答的一致性判断其是否具备内部知识 [34, 40]。如果模型不具备相应的内部知识，那么每次的输出会较为随机，从而导致多次回答的一致性不强；反之，如果模型具备内部知识，其每次都会回答正确的知识、随机性较弱，输出则会有更高的一致性。然而，这种方式同样面临着模型因过度自信而“执拗”地一直给出相同的错误答案的情况。并且，多次询问还需要耗费大量的时间和计算资源，在实际使用中可行性较低。

(2) 观察训练数据

通过询问来判断内部知识的方法存在的模型回答可靠性较低的问题。我们可以转向观察更为可靠的**训练数据**。大语言模型所具备的内部知识来源于其训练数据。因此，如果模型的训练数据中不包含当前问题的相关信息，那么，模型自然也就未曾习得相应的知识。此外，类比人类的学习模式，我们通常对常见或反复学习的知识掌握程度更好，而对低频的知识则较弱。因此，对于训练数据中出现频率很低的知识，模型对它们的学习程度可能也是比较低的，即**知识在训练数据中的出现**

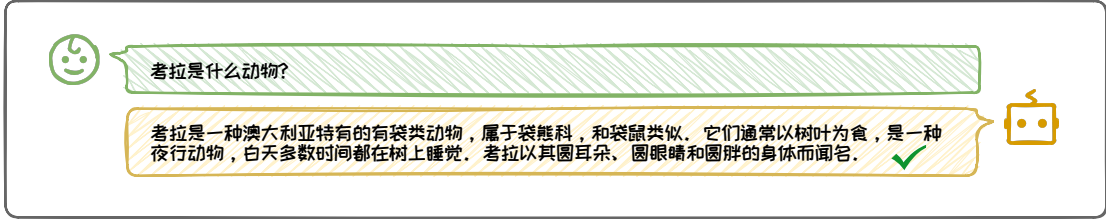


图 6.20: 模型对流行知识的回答示例。

频率与模型对该知识的记忆程度是正相关的 [22]。所以，我们可以通过判断训练数据中是否包含相应的知识来判断模型是否掌握了相应的知识。然而，这种方式存在一定的局限性。首先，由于大语言模型的训练数据规模已经达到了数万亿级别，因此这种统计的手段非常耗时。此外，对于许多商业大语言模型，例如 ChatGPT、GPT4 等，它们的训练数据并不是公开可获取的，在这种情况下，此方案无法执行。

(3) 构造伪训练数据统计量

当训练数据不可获取时，我们可以设计伪训练数统计量来拟合训练数据的相关情况。比如，由于模型对训练数据中低频出现的知识掌握不足，而对更“流行”（高频）的知识掌握更好，因此实体的流行度作可以作为伪训练数据统计量。代表性工作 [33] 利用 **Wikipedia** 的页面浏览量来衡量实体的流行度，浏览量越大，表明该实体越流行，也就更可能被大语言模型所记忆。图 6.20 和图 6.21 分别展示了流行与不流行的知识的例子。从这两个例子中可以发现，对于流行知识**考拉**，模型能够直接给出正确的回答，而对于不流行知识**考拉的基因数量**，模型给出了错误的答案（正确答案应为 26,558）。由此可见，实体的流行度确实一定程度上反映了模型的内部知识，因此，可以通过设定一个流行度阈值来判别模型是否具备相应的内部知识。然而，这种流行度的定义依赖于数据形成时间，并且未必能精准拟合训练数据的分布，因此存在一定的局限性。

2. 内部观测法

为了进一步深入了解大语言模型的内部知识，在模型参数可访问的情况下，可

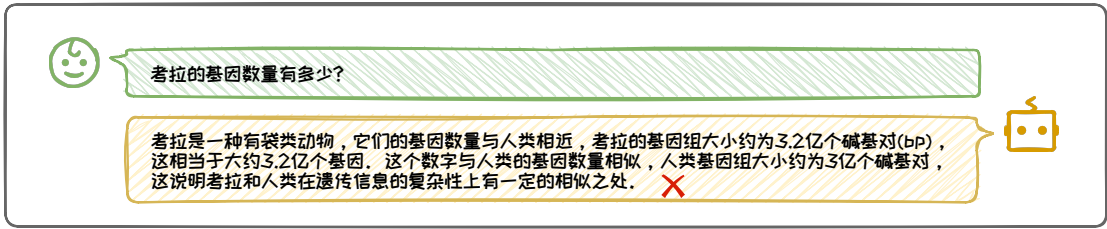


图 6.21: 模型对不流行知识的回答示例。

以通过观测模型内部的隐藏状态来更精确地评估其知识掌握情况。其可以类比人类测谎的过程，科学家通过分析脑电波、脉搏、血压等人类内部状态变化来推断大脑的活动和认知状态，从而判断其是否在说谎或隐藏某些信息。同样地，对于大语言模型，可通过分析模型在生成时每一层的隐藏状态变化，比如注意力模块的输出、多层感知器 (MLP) 层的输出与激活值变化等，来进行评估其内部知识水平。这是因为大语言模型在生成文本时，是对输入序列进行建模和预测，模型内部状态的变化反映了模型对当前上下文理解和下一步预测的确定性。如果模型表现出**较高的内部不确定性**，如注意力分布较为分散、激活值变化较大等，就可能对当前上下文缺乏充分的理解，从而无法做出有把握的预测。

由于模型的内部知识检索主要发生在中间层的前馈网络中 [36]，因此在处理包含或不包含内部知识的不同问题时，模型的中间层会展现出不同的动态变化。基于这一特性，我们可以训练分类器进行判别，这种方法被称为探针。例如，Liang 等人 [30] 针对三种类型的内部隐藏状态设计了探针实验，分别是注意力层输出 (Attention Output)、MLP 层输出 (MLP Output) 和隐层状态 (Hidden States)，如图 6.22 所示。对于每个输入问题，研究者利用训练好的探针，即线性分类器，来根据问题所对应的内部表示预测该问题是属于模型“已知”（即模型具备相关知识）还是“未知”（即模型缺乏相关知识）。结果显示，不同大语言模型在利用中间层的内部表示进行分类时，均能够实现较高的分类准确率。这表明中间层的内部隐藏状态能够有效地反映模型对问题的理解和相关知识储备。

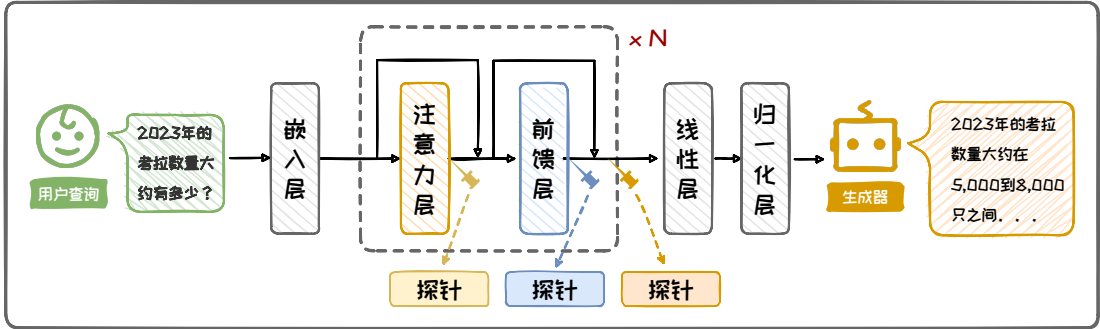


图 6.22: 模型内部状态探针。

然而，这种依赖于内部状态的检测方法也存在一定的局限性，它并不适用于黑盒语言模型，因为我们无法直接访问其内部隐藏状态。另外，模型的输入也需要仔细设计，因为模型有时所展现出的不确定性，可能并非源于对问题的知识缺失，而是问题本身固有的模糊性或歧义性所致。总体而言，基于内部状态评估内部知识的工作目前尚处于初步探索阶段，具体的方案设计有待进一步完善，例如如何构建模型“已知”和“未知”问题的数据集、如何量化内部状态的不确定性、不同内部表示的比对方法，如何设计内部状态检测方案等。我们需要在更广泛的数据集和更多样的模型架构上展开研究，以验证这一方法的有效性和普适性。但是，这是一条充满潜力的新路径，有望为我们从内部视角深入了解大语言模型的知识提供新的视角与方法。

6.4.2 何处增强

在确定大语言模型需要外部知识后，我们需要考虑在何处利用检索到的外部知识，即**何处增强**的问题。得益于大语言模型的上下文学习能力、注意力机制的可扩展性以及自回归生成能力，其输入端、中间层和输出端都可以进行知识融合操作。在输入端，可以将问题和检索到的外部知识拼接在 **Prompt** 中，然后输入给大语言模型；在中间层，可以采用**交叉注意力**将外部知识直接编码到模型的隐藏状

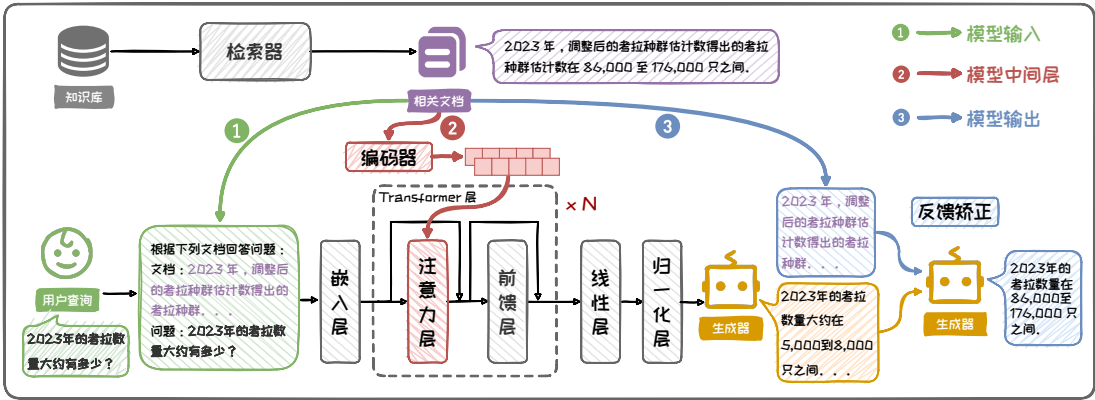


图 6.23: 增强实施位置示意图。

态中；在输出端，可以利用外部知识对生成的文本进行后矫正。图 6.23 通过一个例子展示了上述三种增强位置的实施方案。三种增强位置分别具有不同的优缺点，适合不同的场景。本小节将对其进行逐一介绍。

(1) 在输入端增强

在输入端增强的方法直接将检索到的外部知识文本与用户查询拼接到 Prompt 中，然后输入给大语言模型。其是当前主流的增强方法。此方式的重点在于 Prompt 设计以及检索到的外部知识的排序。良好的 Prompt 设计和外部知识排序，可以使模型更好地理解、利用外部知识。在设计 Prompt 的过程中，可以运用 CoT [54] 等 Prompt 技巧，具体方法可参阅本书第三章的 Prompt 工程内容。

在输入端增强的方法直观且易于实现。模型可以直接从输入的上下文中提取到所需信息，无需复杂的处理或转换。然而，当检索到的文本过长时，可能导致输入序列过长，甚至超出模型的最大序列长度限制。这给模型的上下文理解带来挑战，并且还会增加模型推理计算成本、增加其计算负担。这种方法对大语言模型的长文本处理能力和上下文理解能力要求较高。

(2) 在中间层增强

在中间层增强增强的方法利用注意力机制的灵活性，先将检索到的外部知识转换为向量表示，然后将这些向量插入通过交叉注意力融合到模型的隐藏状态中。

在第6.2节中介绍的 Retro [6] 方法，就是采用这种方式的典型代表。这种方法能够更深入地影响模型的内部表示，可能有助于模型更好地理解 and 利用外部知识。同时，由于向量表示通常比原始文本更为紧凑，这种方法可以减少对模型输入长度的依赖。然而，这种方法需要对模型的结构进行复杂的设计和调整，无法应用于黑盒模型。

(3) 在输出端增强

在输出端增强的方法利用检索到的外部知识对大语言模型生成的文本进行校准，是一种后处理的方法。在此类方法中，模型首先在无外部知识的情况下生成一个初步回答，然后再利用检索到的外部知识来验证或校准这一答案。校验过程基于生成文本与检索文本的知识一致性对输出进行矫正。矫正可以通过将初步回答与检索到的信息提供给大模型，让大模型检查并调整生成的回答来完成。例如，Yu 等人提出的 REFEED 框架 [59] 是此类方法典型代表。这种方法的优点是可以确保生成的文本与外部知识保持一致，提高答案的准确性和可靠性。然而，其效果在很大程度上依赖于检索到的外部知识的质量和相关性。若检索到的文档不准确或不相关，则会导致错误的校准结果。

综上，上述三种增强方式各有优劣，在实际应用中，我们可以根据具体的场景和需求，灵活选择不同的方案。并且，由于上述三种方案是相互独立的，它们也可以组合使用，以实现更优的增强效果。

6.4.3 多次增强

在实际应用中，用户对大语言模型的提问可能是复杂或模糊的。复杂问题往往涉及多个知识点，需要多跳（multi-hop）的理解；而模糊问题往往指代范围不明，难以一次就理解问题的含义。对于复杂问题和模糊问题，我们难以通过一次检索增强就确保生成正确，多次迭代检索增强在所难免。处理复杂问题时，常采用分解式

增强的方案。该方案将复杂问题分解为多个子问题，子问题间进行迭代检索增强，最终得到正确答案。处理模糊问题时，常采用**渐进式增强**的方案。该方案将问题的不断细化，然后分别对细化的问题进行检索增强，力求给出全面的答案，以覆盖用户需要的答案。本小节将对这两种方案展开介绍。

1. 分解式增强

复杂问题通常包含多个知识点。例如，在“世界上睡眠时间最长的动物爱吃什么？”这个问题中，包含着“世界上睡眠时间最长的动物是什么？”和“这个动物爱吃什么？”两个知识点。对复杂问题进行作答，需要多跳理解。因此，在复杂问题的检索增强中，我们通常无法仅通过一次检索增强就得到满意的答案。在这种情况下，模型可以将多跳问题分解为一个个子问题，然后在子问题间迭代地进行检索增强，最后得出正确结论，即分解式增强。

DEMONSTRATE-SEARCH-PREDICT (DSP) [25] 是一种具有代表性的分解式增强框架。该框架主要包含以下三个模块：(1) **DEMONSTRATE** 模块，通过上下文学习的方法，将复杂问题分解为子问题；(2) **SEARCH** 模块，对子问题进行迭代检索增强，为最终决策提供综合的外部知识；(3) **PREDICT** 模块，根据 **SEARCH** 提供的综合外部知识生成最终回答。

如图 6.24 所示，以“世界上睡眠时间最长的动物爱吃什么？”为例对 DSP 的流程进行介绍。首先，**DEMONSTRATE** 模块会在训练集中寻找类似问题的示例，以此来演示问题如何被分解。这些示例将指导大语言模型生成针对子问题的精确查询。例如，第一个子问题是：“世界上睡眠时间最长的动物是什么？”。随后，**SEARCH** 模块将利用第一个子问题来检索相关信息，从而确定睡眠最长的动物是考拉，并基于这一新信息形成第二个子问题：“考拉爱吃什么？”，随后再次执行检索，以找到描述考拉饮食习惯的相关文本。最终，在 **PREDICT** 模块中，大语言模型将综合所有信息得出最终答案，即“世界上睡眠时间最长的动物是考拉，爱吃桉树叶。”

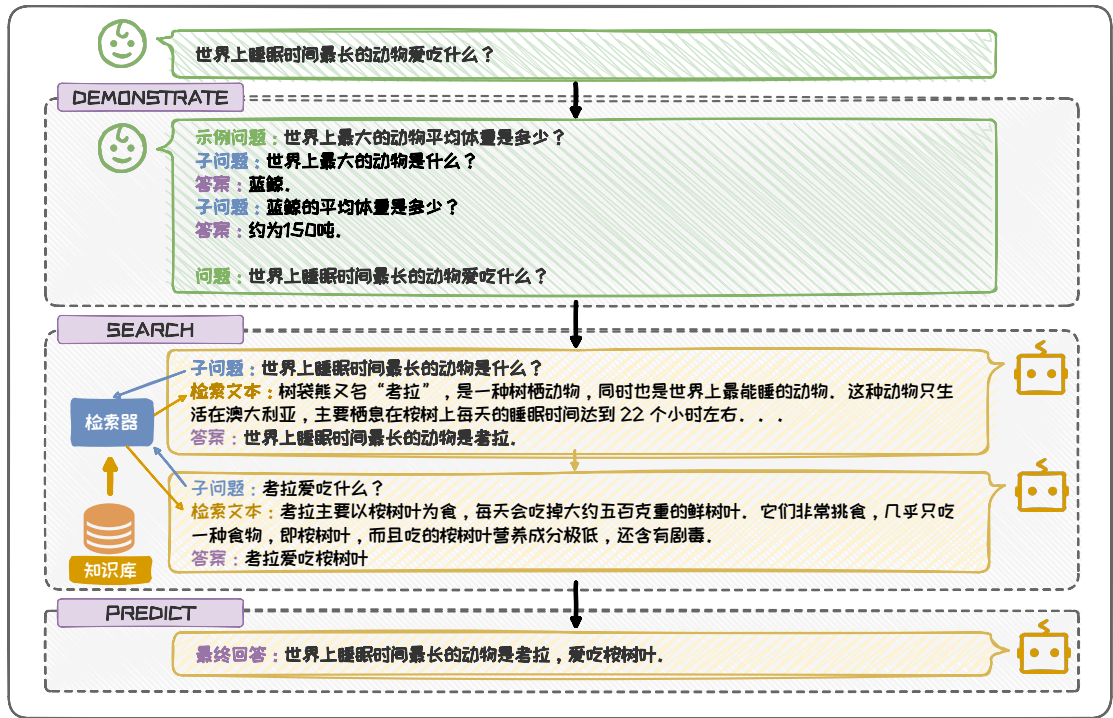


图 6.24: DSP 流程示意图。

分解式增强将复杂问题化整为零，降低了单次检索增强的难度。其性能很大程度上取决于子问题分解的质量。不同领域的复杂问题可能有着不同的分解范式，因此常常需要根据具体任务对问题分解方案进行设计。

2. 渐进式增强

在模糊问题中，问题主体通常指代不明，容易引发歧义。例如，在“国宝动物爱吃什么？”这个问题中，由于不同国家国宝动物不同，且部分国家的国宝动物不止一种，因此我们无法确定询问的是其中的哪一种动物。在处理这样的模糊问题时，我们可以对问题进行渐进式地拆解、细化，然后对细化后的问题进行检索，利用检索到的信息增强大模型。这种渐进式的增强方法可以帮助大语言模型掌握更全面的信息。

TREE OF CLARIFICATIONS (TOC) [26] 是渐进式增强的代表性框架。该框架通过递归式检索来引导大语言模型在**树状结构**中探索给定模糊问题的多种澄

清路径。图 6.25 展示了 TOC 框架的整个工作流，我们以“国宝动物爱吃什么？”这一问题为例进行说明。TOC 框架首先启动第一轮检索，根据检索到的相关文档和原始问题，生成一系列具体的细化问题，例如：“中国的国宝动物爱吃什么？”，“澳洲的国宝动物爱吃什么？”。在此过程中，框架会根据细化问题与原问题的相关性 & 知识一致性进行剪枝。随后，针对每个细化问题，框架独立展开深入的检索并对问题进一步细化，例如：“澳洲的国宝动物考拉爱吃什么？”。最终，我们能够构建出多条完整的知识路径，每条路径的末端（叶节点）都代表了对原始问题的不同但是有效的解答。通过整合所有有效的叶节点，我们能够得出一个精确且全面的长回答，例如对此问题我们得到：“中国的国宝动物熊猫爱吃竹子；澳洲的国宝动物考拉爱吃桉树叶；澳洲的国宝动物袋鼠爱吃杂草和灌木……”

这种渐进式的检索方法能够显著改善大语言模型对模糊问题的生成效果。然而，其需要进行多轮检索并生成多个答案路径，整个系统的计算量会随着问题复杂度呈指数级增长。此外，多轮的检索与生成不仅会带来显著的时间延迟，多个推理路径还为推理带来不稳定性，容易引发错误。

6.4.4 降本增效

检索出的外部知识通常包含大量原始文本。将其通过 Prompt 输入给大语言模型时，会大幅度增加输入 Token 的数量，从而增加了大语言模型的推理计算成本。此问题可从去除冗余文本与复用计算结果两个角度进行解决。本小节将对这两个角度分别展开介绍。

1. 去除冗余文本

在 RAG 中，检索出的原始文本通常包含大量的无益于增强生成的冗余信息。这些冗余信息不仅增加了输入 Token 的长度，而且还有可能对大模型产生干扰，导致生成错误答案。去除冗余文本的方法通过对检索出的原始文本的词句进行过

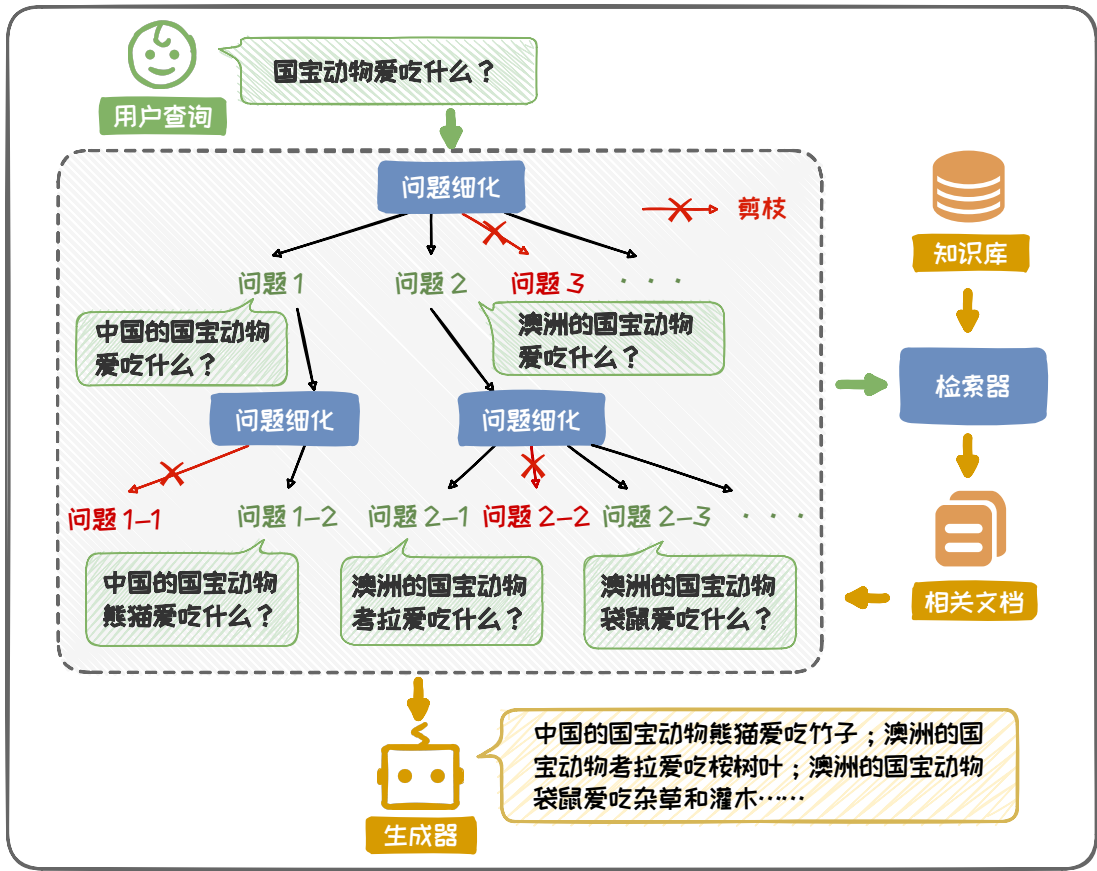


图 6.25: TOC 框架流程示意图。

滤，从中选择出部分有益于增强生成的部分。去除冗余文本的方法主要分为三类：Token 级别的方法，子文本级别的方法以及全文本级别的方法。这些方法通过不同的机制来筛选和优化检索出的原始文本，以减少无益信息，确保生成内容的相关性和准确性。以下分别对这三类方法分别展开介绍。

Token 级别的方法通过对 Token 进行评估，对文本中不必要的 Token 进行剔除。困惑度是判断 Token 重要性的重要指标。直观上说，如果一个 Token 的困惑度低，这意味着模型有很高的概率预测到这个 Token，表明该 Token 易于预测且较为普遍，因此它携带的新信息量较少，可能是冗余的；反之，如果一个 Token 的困惑度高，则表明这个 Token 携带了更多的信息量。LongLLMLingua [20] 框架利用小模型评

估 Token 的困惑度，然后基于困惑度删除冗余文本。其首先进行问题感知的**粗粒度压缩**，即在给定问题条件下通过计算文档中所有 Token 困惑度的均值来评估文档的重要性，困惑度越高表示文档信息量越大。随后，执行问题感知的**细粒度压缩**，即进一步计算文档中每个 Token 的困惑度并去除其中低困惑度的 Token。此外，论文还引入了**文档重排序机制**、**动态压缩比率**以及**子序列恢复机制**，确保重要文档和文档中的重要信息被有效利用。

子文本级别的方法通过对子文本进行打分，对不必要的子文本成片删除。**FIT-RAG** [35] 是子文本级别方法中的代表性方法。该方法中采用了预先训练的双标签子文档打分器，从两个维度评估文档的有用性：一是**事实性**，即文档是否包含查询的答案信息；二是**模型的偏好程度**，即文档是否易于模型理解。对于检索到的文档，首先利用滑动窗口将其分割成多个子文档，然后使用双标签子文档打分器对这些子文档分别进行评分。最后，删除掉评分较低子文档，从而有效地去除冗余文本。

全文本级别的方法直接从整个文档中抽取出重要信息，以去除掉冗余信息。经典方法 **PRCA** [57] 通过训练能够提炼重点内容的信息提取器对文本中的重要信息进行提取。该方法分为两个阶段，上下文提取阶段与奖励驱动阶段。在**上下文提取阶段**，以最小化压缩文本与原输入文档之间的差异为目标，对信息提取器进行监督学习训练，学习如何将输入文档精炼为信息丰富的压缩文本。在**奖励驱动阶段**，大语言模型作为奖励模型，其根据压缩文本生成的答案与真实答案之间的相似度作为奖励信号，通过强化学习对信息提取器进行优化。最终得到的信息提取器可以直接将输入文档转化为压缩文本，端到端地去除冗余文本。

2. 复用计算结果

除了对冗余信息进行筛除，我们还可以对计算必需的中间结果进行复用，以优化 RAG 效率。在大语言模型进行推理的自回归过程中，每个 Token 都需要用到

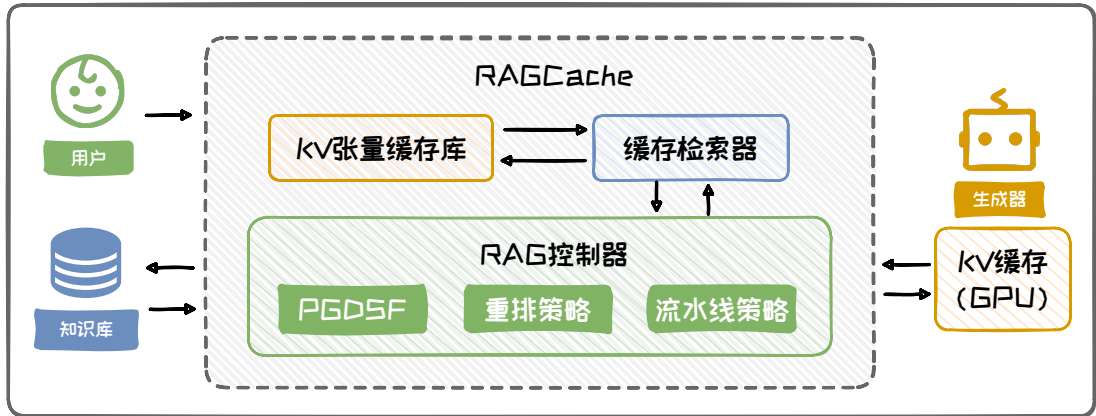


图 6.26: RAGCache 框架流程示意图。

之前 Token 注意力模块涉及的 Key 和 Value 的结果。为了避免对每个 Token 都重新计算前面的 Key 和 Value 的结果，我们可以将之前计算的 Key 和 Value 的结果进行缓存（即 KV-cache），在需要是直接从 KV-cache 中调用相关结果，从而避免重复计算。但是，随着输入文本长度的增加，KV-cache 的 GPU 显存占用会随之剧增，甚至会远远超过模型参数所占用的显存大小 [46]。然而，RAG 的输入通常包含检索出来的文本，导致输入文本很长，导致 RAG 中的 KV-cache 存储成本高昂。

不过，在 RAG 中，不同用户查询经常检索到相同的文本，而且常见的查询通常数量有限。因此，我们可以将常用的重复文本的 KV-cache 进行复用，避免每次查询都对其进行计算，以降低存储成本。基于此，RAGCache [21] 设计了一种 RAG 系统专用的多级动态缓存机制，如图 6.26 所示。RAGCache 系统由三个核心部分组成：KV 张量缓存库、缓存检索器与 RAG 控制器。其中，KV 张量缓存库采用树结构来缓存所计算出的文档 KV 张量，其中每个树节点代表一个文档；缓存检索器则负责在缓存库中快速查找是否存在所需的缓存节点；而 RAG 控制器作为系统的策略中枢，负责制定核心的缓存策略。具体而言，RAG 控制器首先采用了一种前缀感知的贪婪双重尺度频率（PGDSF）替换策略。该策略综合考虑了文档节点的访问频率、大小、访问成本以及最近访问时间，使得频繁使用的文档能够被快速检索

到。随后，**重排策略**通过调整请求的处理顺序，优先处理那些能够更多利用缓存数据的请求，以减少重新计算的需求。这样的请求在队列中享有较高的优先级，从而优化了资源使用。最后，系统还引入了**动态推测流水线策略**，通过并行 KV 张量检索和模型推理的步骤，有效减少了端到端延迟。

综上，通过结合上述**输入 Prompt 压缩**与**KV-cache 机制**，RAG 框架可以在保持高性能的同时，显著提升其效率。这不仅有助于在资源受限的环境中部署模型，还可以提高模型在实际应用中的响应速度。

6.5 实践与应用

通过引入外部知识，RAG 可以有效的缓解大语言模型的幻觉现象，拓展了大语言模型的知识边界。其优越的性能使引起了广泛关注，成为炙手可热的前沿技术，并在众多应用场景中落地。在本节中，我们将探讨搭建简单 RAG 系统的方法，以及 RAG 的两类典型应用。

6.5.1 搭建简单 RAG 系统

为了助力开发者们高效且便捷地构建 RAG 系统，当前已有诸多成熟开源框架可供选择，其中最具代表性的便是 LangChain⁸与 LlamaIndex⁹。这些框架提供了一系列完备的工具与接口，使得开发者们能够轻松地将 RAG 系统集成到他们的应用中，例如聊天机器人、智能体（Agent）等。接下来，我们将首先简要概述这两个框架的特色及其核心功能，然后讲解如何利用 LangChain 来搭建一个简单的 RAG 系统。

⁸<https://www.langchain.com>

⁹<https://www.llamaindex.ai>

1. LangChain 与 LlamaIndex

(1) LangChain

LangChain 旨在简化利用大语言模型开发应用程序的整个过程。它提供了一系列模块化的组件，帮助开发者部署基于大语言模型的应用，其中就包括 RAG 框架的构建。LangChain 主要包含六大模块：**Model IO**、**Retrieval**、**Chains**、**Memory**、**Agents** 和 **Callbacks**。其中 Model IO 模块包含了各种大模型的接口以及 Prompt 设计组件，Retrieval 模块包含了构建 RAG 系统所需要的核心组件，包括**文档加载**、**文本分割**、**向量构建**，**索引生成以及向量检索**等，还提供了非结构化数据库的接口。而 Chains 模块可以将各个模块链接在一起逐个执行，Memory 模块则可以存储对话过程中的数据。此外，Agent 模块可以利用大语言模型来自动决定执行哪些操作，Callback 模块则可以帮助开发者干预和监控各个阶段。总体而言，**LangChain 提供了一个较为全面的模块支持**，帮助开发者们轻松便捷地构建自己的 RAG 应用框架。

(2) LlamaIndex

与 LangChain 相比，LlamaIndex 更加专注于**数据索引与检索**的部分。这一特性使得开发者能够**迅速构建高效的检索系统**。LlamaIndex 具备从多种数据源（如 API、PDF 文件、SQL 数据库等）中提取数据的能力，并提供了一系列高效的工具来对这些数据进行向量化处理和索引构建。在数据查询方面，LlamaIndex 同样提供了高效的检索机制。此外，在获取到上下文信息后，LlamaIndex 还支持对这些信息进行过滤、重新排序等精细化操作。值得一提的是，LlamaIndex 框架还能够与 LangChain 框架相结合，从而实现更加多样化的功能。总体而言，**LlamaIndex 侧重于索引与检索，在查询效率上的表现更为突出**，非常适用于在大数据量的场景下构建更为高效的 RAG 系统。

2. 基于 LangChain 搭建简单 RAG 系统

本小节将以 LangChain 框架为例，参考其官方文档¹⁰，演示如何快速搭建一套简单的 RAG 系统。

1) 安装与配置：首先，需要在环境中安装 LangChain 框架及其依赖项。

```
# 安装LangChain框架及其依赖项
!pip install langchain langchain_community langchain_chroma
```

2) 数据准备与索引构建：接下来，我们需要准备数据并构建索引。LangChain 的 DocumentLoaders 中提供了种类丰富的文档加载器，例如，我们可以使用 WebBaseLoader 从网页中加载内容并将其解析为文本。

```
from langchain_community.document_loaders import WebBaseLoader
# 使用WebBaseLoader加载网页内容：
loader = WebBaseLoader("https://example.com/page")
docs = loader.load()
```

加载完成后，由于加载的文档可能过长，不适合模型的上下文窗口，需要将文档分割成合适的大小。LangChain 提供了 TextSplitter 组件来实现文档分割。

```
from langchain_text_splitters import RecursiveCharacterTextSplitter
# 使用TextSplitter将长文档分割成更小的块，其中chunk_size表示分割文档的长度，
  chunk_overlap表示分割文档间的重叠长度
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
splits = text_splitter.split_documents(docs)
```

接下来我们需要对分割后的文本块进行索引化，以便后续进行检索。这里我们可以调用 Chroma 向量存储模块和 OpenAIEmbeddings 模型来存储和编码文档。

¹⁰<https://python.langchain.com/v0.2/docs/tutorials/rag/>

```
from langchain_chroma import Chroma
from langchain_openai import OpenAIEmbeddings
# 使用向量存储(如Chroma)和嵌入模型来编码和存储分割后的文档
vectorstore = Chroma.from_documents(documents=splits, embedding=OpenAIEmbeddings
    ())
```

3) RAG 系统构建：在构建好知识源之后，接下来可以开始构建一个基础的 RAG 系统。该系统包括检索器与生成器两部分，具体工作流程如下：对于用户输入的问题，检索器首先搜索与该问题相关的文档，接着将检索到的文档与初始问题一起传递给生成器，即大语言模型，最后将模型生成的答案返回给用户。

首先进行检索器的构建，这里我们可以基于 `VectorStoreRetriever` 构建一个 `Retriever` 对象，利用向量相似性进行检索。

```
# 创建检索器
retriever = vectorstore.as_retriever()
```

接下来是生成器部分的构建，这里我们可以使用 `ChatOpenAI` 系统模型作为生成器。在这一步骤中，需要设置 OpenAI 的 API 密钥，并指定要使用的具体模型型号。例如，我们可以选择使用 `gpt-3.5-turbo-0125` 模型。

```
import os
os.environ["OPENAI_API_KEY"] = 'xxx'
from langchain_openai import ChatOpenAI
llm = ChatOpenAI(model="gpt-3.5-turbo-0125")
```

随后是输入 Prompt 的设置，LangChain 的 Prompt Hub 中提供了多种预设的 Prompt 模板，适用于不同的任务和场景。这里我们选择一个适用于 RAG 任务的 Prompt。

```
from langchain import hub

# 设置提示模板

prompt = hub.pull("rlm/rag-prompt")
```

最后我们需要整合检索与生成,这里可以使用 LangChain 表达式语言(LangChain Execution Language, LCEL) 来方便快捷地构建一个链,将检索到的文档、构建的输入 Prompt 以及模型的输出组合起来。

```
from langchain_core.runnables import RunnablePassthrough
from langchain_openai import ChatOpenAI
from langchain_core.output_parsers import StrOutputParser

# 使用LCEL构建RAG链
rag_chain = (
    {"context": retriever | format_docs, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)

# 定义文档格式化函数
def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)

# 使用RAG链回答问题
response = rag_chain.invoke("What is Task Decomposition?")
print(response)
```

通过以上步骤,我们可以方便快捷地使用 LangChain 迅速搭建一个基础 RAG 系统。LangChain 提供了一系列强大的工具和组件,使得构建和整合检索与生成过程变得简单而高效。

6.5.2 RAG 的典型应用

本小节将介绍 RAG 系统的两个典型应用案例：(1) 智能体 (Agent)；(2) 垂域多模态模型增强。

1. 智能体 (Agent)

RAG 在 Agent 系统中扮演着重要角色。在 Agent 系统主动规划和调用各种工具的过程中，需要检索并整合多样化的信息资源，以更精确地满足用户的需求。RAG 通过提供所需的信息支持，助力 Agent 在处理复杂问题时展现出更好的性能。图 6.27 展示了一个经典的 Agent 框架 [50]，该框架主要由四大部分组成：配置 (Profile)、记忆 (Memory)、计划 (Planning) 和行动 (Action)。其中记忆模块、计划模块与行动模块均融入了 RAG 技术，以提升整体性能。

具体而言，(1) 配置模块通过设定基本信息来定义 Agent 的角色，这些信息可以包括 Agent 的年龄、性别、职业等基本属性，以及反映其个性和社交关系的信息；(2) 记忆模块存储从环境中学习到的知识以及历史信息，支持记忆检索、记忆更新和记忆反思等操作，允许 Agent 不断获取、积累和利用知识。在这一模块中，RAG 通过检索相关信息来辅助记忆的读取和更新；(3) 计划模块赋予 Agent 将复杂任务分解为简单的子任务的能力，并根据记忆和行动反馈不断调整。RAG 在此模块中通过提供相关的信息，帮助 Agent 更合理有效地规划任务；(4) 行动模块则负责将 Agent 的计划转化为具体的行动，包括网页检索、工具调用以及多模态输出等，能够对环境或 Agent 自身状态产生影响，或触发新的行动链。在这一模块中，RAG 通过检索相关信息来辅助 Agent 的决策和行动执行。通过这种模块化且集成 RAG 技术的方法，Agent 能够更加高效和智能地响应用户需求，展现出更加卓越的性能。

我们以“我现在在澳大利亚，我想去抱考拉，应该怎么办呢？”这一具体问题

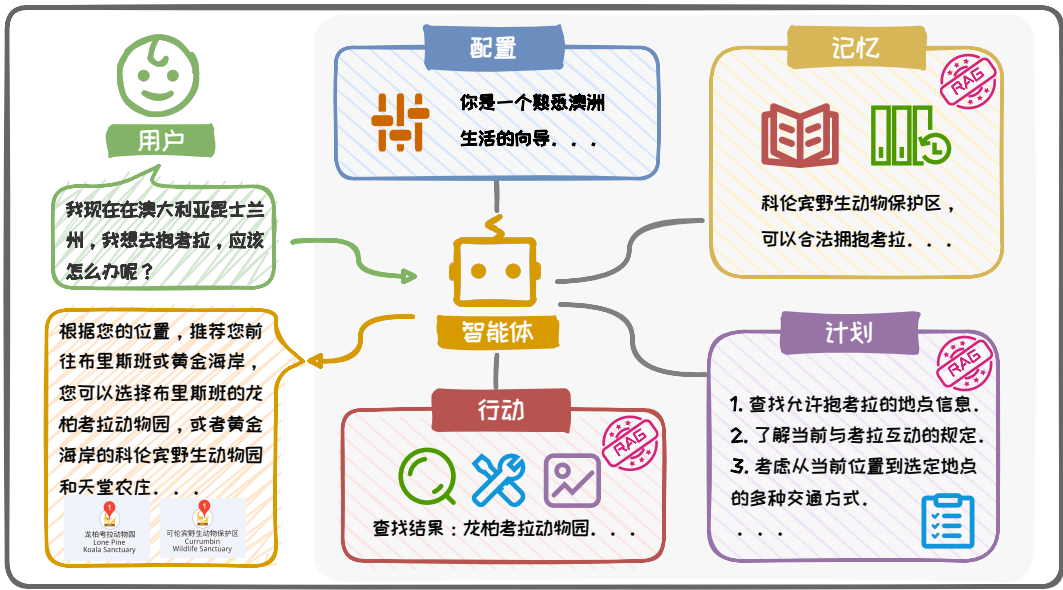


图 6.27: Agent 框架流程示意图。

为例，来展示 Agent 处理问题的流程。在这个例子中，用户提出了一个明确的需求。Agent 将通过以下步骤来完成任务：

- **角色配置**：首先，Agent 利用**配置模块**进行初始化，设定其角色为专业的旅游顾问，明确其任务目标是协助用户实现与考拉亲密接触的愿望；
- **任务规划**：针对用户的需求，**计划模块**规划如何帮助用户实现“抱考拉”的愿望。例如确定当前位置附近存在考拉的地点、了解当地关于与野生动物互动的法律规定、以及前往目的地的方式等。在这一阶段，**RAG** 通过提供相关的旅游景点和法律法规信息，使规划更加精准；
- **信息检索**：**记忆模块**首先在记忆中检索已有的相关信息。如果记忆中不包含所有必要的信息，**行动模块**将激活并调用工具从外部知识源（如旅游网站等）中进行检索。在这一过程中，**RAG** 确保以最高效率检索到与考拉互动最相关的信息；
- **信息整合与决策**：**计划模块**将利用上述检索到的信息制定最佳行动方案。这可能涉及推荐特定的地点、提供出行建议和强调安全须知。**RAG** 在此环节中

整合信息，辅助 Agent 做出明智的决策；

- **信息输出：**最后，**行动模块**将 Agent 确定的行动方案以易于理解的方式输出给用户，包括详细的路线规划以及相关地点的描述、图像等信息。

通过这个例子，我们可以清晰地看到 Agent 框架中的各个模块如何协同工作来完成一个具体的预测任务，其中 RAG 的作用在于快速检索信息并整合知识，为用户提供一个全面而精确的解决方案。

2. 多模态垂直领域应用

在前面的章节中，我们主要聚焦于文本领域的 RAG 系统，而今，在诸多涉及到多模态数据的垂直领域中，RAG 也展现出了广阔的应用前景。例如，在医疗领域中，多模态数据十分普遍，包括 X 光片、MRI、CT 扫描等影像资料，病历、生理监测数据等文本资料。这些数据不仅来源广泛，而且彼此之间存在着复杂的相互联系。因此，在应对这些任务时，RAG 系统必须具备融合与洞察不同模态数据的能力，以确保其精准高效地发挥作用。

目前，已经出现了一些多模态垂直领域的 RAG 应用，例如 Ossowski 等人提出的医学领域多模态检索框架 [38]。图 6.28 展示了该框架的整个工作流程。此处以**给定一张 X 光照片，询问该照片所对应的可能症状**这一任务为例进行说明。该框架首先**提取图像与文本的特征表示**，深入理解图像内容 with 问题语义，为检索模块提供丰富的特征向量。随后，该框架精心设计了一个**多模态检索模块**，利用这些特征向量在医学知识库中进行精准检索，从而获取与输入问题最为相关的信息。这些信息可能包括医学案例、症状描述、潜在病因及治疗方案等，它们以文本和图像的多种形式呈现。最终，**Prompt 构建模块**将这些信息进行整合，辅助模型生成准确的回答。在本例中，模型生成的答案是“心肌肥大”。

除了医疗领域，RAG 在其他垂直领域，如金融 [8]、生物学 [53] 中也展现了其在处理专业信息上的强大能力，显著提升了专业人士的决策效率。在各类任务中，

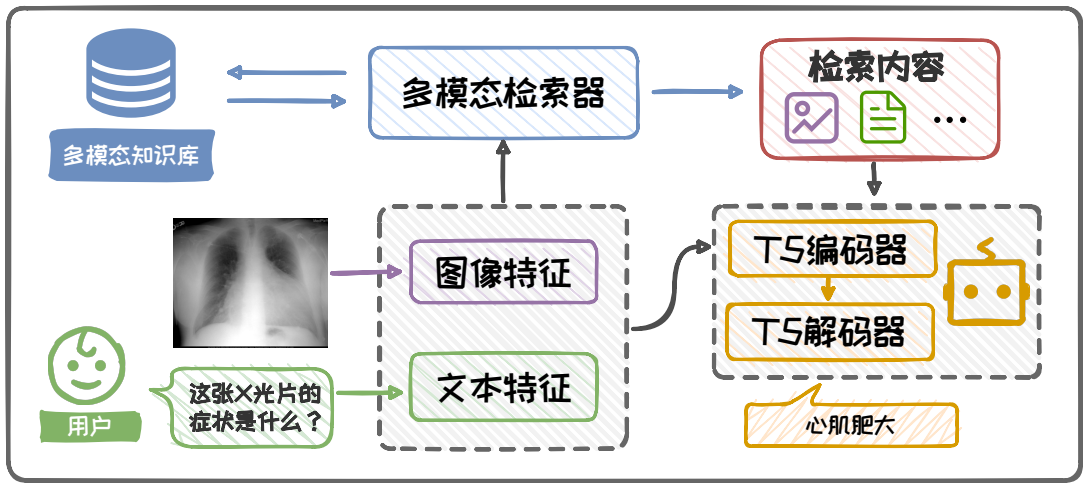


图 6.28: 多模态垂域 RAG 框架示意图。

除了图片信息 [10], RAG 在音频 [7]、视频 [52] 等其他多模态场景中也同样有着出色的表现。随着技术的进步和数据资源的丰富, 我们期待未来 RAG 能够在更多垂直领域、更多数据模态中发挥关键作用。

参考文献

- [1] Josh Achiam et al. "Gpt-4 technical report". In: *arXiv preprint arXiv:2303.08774* (2023).
- [2] Akiko Aizawa. "An information-theoretic perspective of tf-idf measures". In: *IPM*. 2003.
- [3] Akari Asai et al. "Self-rag: Learning to retrieve, generate, and critique through self-reflection". In: *arXiv preprint arXiv:2310.11511* (2023).
- [4] Jon Louis Bentley. "Multidimensional binary search trees used for associative searching". In: *Communications of the ACM* (1975).
- [5] Sebastian Borgeaud et al. "Improving language models by retrieving from trillions of tokens". In: *ICML*. 2022.
- [6] Sebastian Borgeaud et al. "Improving language models by retrieving from trillions of tokens". In: *ICML*. 2022.

- [7] David M Chan et al. “Using external off-policy speech-to-text mappings in contextual end-to-end automated speech recognition”. In: *arXiv preprint arXiv:2301.02736* (2023).
- [8] Jian Chen et al. “FinTextQA: A Dataset for Long-form Financial Question Answering”. In: *arXiv preprint arXiv:2405.09980* (2024).
- [9] Jiangui Chen et al. “A Unified Generative Retriever for Knowledge-Intensive Language Tasks via Prompt Learning”. In: *SIGIR*. 2023.
- [10] Wenhui Chen et al. “Re-imagen: Retrieval-augmented text-to-image generator”. In: *arXiv preprint arXiv:2209.14491* (2022).
- [11] Mayur Datar et al. “Locality-sensitive hashing scheme based on p-stable distributions”. In: *SoCG*. 2004.
- [12] Hervé Déjean, Stéphane Clinchant, and Thibault Formal. “A Thorough Comparison of Cross-Encoders and LLMs for Reranking SPLADE”. In: *arXiv preprint arXiv:2403.10407* (2024).
- [13] Mohamad Dolatshah, Ali Hadian, and Behrouz Minaei-Bidgoli. “Ball*-tree: Efficient spatial indexing for constrained nearest-neighbor search in metric spaces”. In: *arXiv preprint arXiv:1511.00628* (2015).
- [14] Paulo Finardi et al. “The Chronicles of RAG: The Retriever, the Chunk and the Generator”. In: *arXiv preprint arXiv:2401.07883* (2024).
- [15] Tiezheng Ge et al. “Optimized product quantization for approximate nearest neighbor search”. In: *CVPR*. 2013.
- [16] Samuel Humeau et al. “Poly-encoders: Transformer architectures and pre-training strategies for fast and accurate multi-sentence scoring”. In: *arXiv preprint arXiv:1905.01969* (2019).
- [17] Gautier Izacard et al. “Atlas: Few-shot learning with retrieval augmented language models”. In: *Journal of Machine Learning Research* (2023).
- [18] Herve Jegou, Matthijs Douze, and Cordelia Schmid. “Product quantization for nearest neighbor search”. In: *IEEE transactions on pattern analysis and machine intelligence* 33.1 (2010), pp. 117–128.
- [19] Hervé Jégou et al. “Searching in one billion vectors: re-rank with source coding”. In: *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2011, pp. 861–864.

- [20] Huiqiang Jiang et al. “Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression”. In: *arXiv preprint arXiv:2310.06839* (2023).
- [21] Chao Jin et al. “RAGCache: Efficient Knowledge Caching for Retrieval-Augmented Generation”. In: *arXiv preprint arXiv:2404.12457* (2024).
- [22] Nikhil Kandpal et al. “Large language models struggle to learn long-tail knowledge”. In: *ICML*. 2023.
- [23] Vladimir Karpukhin et al. “Dense passage retrieval for open-domain question answering”. In: *arXiv preprint arXiv:2004.04906* (2020).
- [24] Omar Khattab and Matei Zaharia. “Colbert: Efficient and effective passage search via contextualized late interaction over bert”. In: *SIGIR*. 2020.
- [25] Omar Khattab et al. “Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp”. In: *arXiv preprint arXiv:2212.14024* (2022).
- [26] Gangwoo Kim et al. “Tree of clarifications: Answering ambiguous questions with retrieval-augmented large language models”. In: *EMNLP*. 2023.
- [27] Mike Lewis et al. “Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension”. In: *arXiv preprint arXiv:1910.13461* (2019).
- [28] Patrick Lewis et al. “Retrieval-augmented generation for knowledge-intensive nlp tasks”. In: *NeurIPS*. 2020.
- [29] Xiaoxi Li et al. “From matching to generation: A survey on generative information retrieval”. In: *arXiv preprint arXiv:2404.14851* (2024).
- [30] Yuxin Liang et al. “Learning to trust your feelings: Leveraging self-awareness in llms for hallucination mitigation”. In: *arXiv preprint arXiv:2401.15449* (2024).
- [31] Yu A Malkov and Dmitry A Yashunin. “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs”. In: *IEEE transactions on pattern analysis and machine intelligence* (2018).
- [32] Yury Malkov et al. “Approximate nearest neighbor algorithm based on navigable small world graphs”. In: *Information Systems* 45 (2014), pp. 61–68.
- [33] Alex Mallen et al. “When not to trust language models: Investigating effectiveness of parametric and non-parametric memories”. In: *ACL*. 2023.

- [34] Potsawee Manakul, Adian Liusie, and Mark JF Gales. “Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models”. In: *EMNLP*. 2023.
- [35] Yuren Mao et al. “FIT-RAG: Black-Box RAG with Factual Information and Token Reduction”. In: *ACM Transactions on Information Systems* (2024).
- [36] Kevin Meng et al. “Locating and editing factual associations in GPT”. In: *NeurIPS*. 2022.
- [37] Stanislav Morozov and Artem Babenko. “Non-metric similarity graphs for maximum inner product search”. In: *Advances in Neural Information Processing Systems* 31 (2018).
- [38] Timothy Ossowski and Junjie Hu. “Multimodal prompt retrieval for generative visual question answering”. In: *ACL*. 2023.
- [39] James Jie Pan, Jianguo Wang, and Guoliang Li. “Survey of vector database management systems”. In: *arXiv preprint arXiv:2310.14021* (2023).
- [40] Ella Rabinovich et al. “Predicting Question-Answering Performance of Large Language Models through Semantic Consistency”. In: *arXiv preprint arXiv:2311.01152* (2023).
- [41] Colin Raffel et al. “Exploring the limits of transfer learning with a unified text-to-text transformer”. In: *The Journal of Machine Learning Research* (2020).
- [42] Ori Ram et al. “In-context retrieval-augmented language models”. In: *Transactions of the Association for Computational Linguistics* (2023).
- [43] Stephen Robertson, Hugo Zaragoza, et al. “The probabilistic relevance framework: BM25 and beyond”. In: *Foundations and Trends® in Information Retrieval* (2009).
- [44] Ferdinand Schlatt, Maik Fröbe, and Matthias Hagen. “Investigating the Effects of Sparse Attention on Cross-Encoders”. In: *ECIR*. 2024.
- [45] Weijia Shi et al. “Replug: Retrieval-augmented black-box language models”. In: *arXiv preprint arXiv:2301.12652* (2023).
- [46] Hanshi Sun et al. “Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding”. In: *arXiv preprint arXiv:2404.11912* (2024).
- [47] Weiwei Sun et al. “Is chatgpt good at search? investigating large language models as re-ranking agent”. In: *arXiv preprint arXiv:2304.09542* (2023).
- [48] Yi Tay et al. “Transformer memory as a differentiable search index”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 21831–21843.

- [49] Hugo Touvron et al. “Llama 2: Open foundation and fine-tuned chat models”. In: *arXiv preprint arXiv:2307.09288* (2023).
- [50] Lei Wang et al. “A survey on large language model based autonomous agents”. In: *Frontiers of Computer Science* 18 (2024), p. 186345.
- [51] Yujing Wang et al. “A Neural Corpus Indexer for Document Retrieval”. In: *ArXiv abs/2206.02743* (2022). URL: <https://api.semanticscholar.org/CorpusID:249395549>.
- [52] Zhenhailong Wang et al. “Language models with image descriptors are strong few-shot video-language learners”. In: *NeurIPS*. 2022.
- [53] Zichao Wang et al. “Retrieval-based controllable molecule generation”. In: *ICLR*. 2022.
- [54] Jason Wei et al. “Chain of Thought Prompting Elicits Reasoning in Large Language Models”. In: *NeurIPS*. 2022.
- [55] David H Wolpert and William G Macready. “No free lunch theorems for optimization”. In: *IEEE transactions on evolutionary computation* 1 (1997), pp. 67–82.
- [56] Mingrui Wu and Sheng Cao. “LLM-Augmented Retrieval: Enhancing Retrieval Models Through Language Models and Doc-Level Embedding”. In: *arXiv preprint arXiv:2404.05825* (2024).
- [57] Haoyan Yang et al. “Prca: Fitting black-box large language models for retrieval question answering via pluggable reward-driven contextual adapter”. In: *EMNLP*. 2023.
- [58] Wenhao Yu et al. “Generate rather than retrieve: Large language models are strong context generators”. In: *ICLR*. 2023.
- [59] Wenhao Yu et al. “Improving Language Models via Plug-and-Play Retrieval Feedback”. In: *arXiv preprint arXiv:2305.14002* (2023).
- [60] Zichun Yu et al. “Augmentation-Adapted Retriever Improves Generalization of Language Models as Generic Plug-In”. In: *arXiv preprint arXiv:2305.17331* (2023).
- [61] Yujia Zhou et al. “DynamicRetriever: A Pre-training Model-based IR System with Neither Sparse nor Dense Index”. In: *ArXiv abs/2203.00537* (2022). URL: <https://api.semanticscholar.org/CorpusID:247187834>.