一、 总体设计

1.1.1 设计原则

1.1.2 系统组成

1.1.3 业务架构

1.1.4 系统架构

从软件架构角度看，RAG 系统的核心功能是进行数据处理，具有典型的数据流（Dataflow）架构风格。针对这一特点，参考目前主流的 RAG 系统架构思路，本方案设计了一个模块化的 RAG 系统架构，整个系统分为基础设施、

包含子系统（Sub-system）、模块（Module）和操作符（Operator）三个层次，其中子系统表示 RAG 过程中一个独立的阶段或过程，每个子系统由多个模块组成。模块指的是一个子系统中需要独立完成的任务，每个任务可能有多种实现方案，每种实现方案由一个操作符来定义。操作符是整个 RAG 中的最小单元，通常是一个独立的算法，有明确的输入和输出，一个模块中的多个操作符可能存在冲突，需要独立使用，也有可能互相协作，产生更大的价值。在三层结构之上，系统支持可以根据应用场景的特点自由组合模块和操作符，以达成更好的效果。

一个典型的知识问答系统可以分为数据准备阶段和检索增强生成两个阶段，如图？所示。

从图中可以看出，在数据准备阶段，首先需要通过数据结构化框架，将数据库、文档、音频、视频、图像等多种类型的原始数据转化为统一的格式，然后根据某种分块原则将数据分割为一定大小的块，最后对每个块进行嵌入处理，存入向量数据库中。

在检索增强生成阶段，用户的查询问题首先通过嵌入模块将其表达为向量，然后通过检索模块在向量数据库中匹配到相关的素材，然后通过基于大模型的生成模块对素材进行改写，最终向用户返回最终结果。

在实际应用中，会加入一些额外的处理模块，对整个过程进行扩展，并引入各种复杂的控制结构，下面将分别进行介绍。

1.1.4.1 顺序（Sequential）结构

RAG 系统顺序结构将整个处理过程的相关模块和运算符组织成线性流水线（pipeline），如图？所示。如果这个流水线中包括预检索和后检索等模块，则表示典型的高级 RAG；否则，它体现了典型的简单 RAG 范式。

目前最广泛使用的 RAG Pipeline 是顺序的，通常在检索之前包括查询重写或 HyDE，并在检

重写-检索-读取（RRR）也是一种典型的顺序结构。查询重写模块是一个较小的可训练语言模型，在强化学习的背景下，重写器的优化被形式化为马尔可夫决策过程，其中 LLM 的最终输出作为奖励。检索器使用了一种稀疏编码模型，BM25。

1.1.4.2 条件（Conditional）结构

RAG 流程与条件结构涉及根据不同条件选择不同的 RAG 路径。通常，这是通过一个路由模块来实现的，该模块根据查询关键词或语义来确定路由。

根据问题的类型选择不同的路线，针对特定情景进行不同的流程。例如，当用户询问严肃问题、政治问题或娱乐话题时，对于大型模型的回答容忍度不同。不同的路由分支通常在检索来源、检索过程、配置、模型和提示方面有所不同。

1.1.4.3 分支（Branching）结构

RAG 流程具有分支结构，与条件方法不同的是，它涉及多个并行分支，而不是在条件方法中从多个选项中选择一个分支。从结构上看，它可以分为两种类型：

- 预检索分支（多查询，并行检索）。这涉及扩展原始查询以获取多个子查询，然后针对每个子查询进行单独的检索。在检索后，该方法允许根据子问题和相应的检索内容立即生成答案。或者，它可能涉及仅使用扩展的检索内容，并将其合并到统一的上下文中进行生成。
- 后检索分支（单一查询，并行生成）。该方法保留原始查询并检索多个文档块。随后，它同时使用原始查询和每个文档块进行生成，最后将生成的结果合并在一起。

REPLUG 体现了一个经典的后检索分支结构，其中预测了每个分支的每个标记的概率。通过加权可能性集成，将不同的分支聚合在一起，并使用最终的生成结果通过反馈来微调检索器，称为 Contriever。

1.1.4.4 循环（Loop）结构

RAG 流程具有循环结构，这是模块化 RAG 的一个重要特征，涉及相互依赖的检索和推理步骤。通常包括用于流程控制的判断模块（judge module）。这可以进一步分为迭代、递归和自适应（主动）检索方法。

1.1.4.5 迭代检索（Iterative Retrieval）结构

有时，单次检索和生成可能无法有效解决需要广泛知识的复杂问题。因此，在 RAG 中可以使用迭代方法，通常涉及固定次数的检索。

迭代检索的一个典型案例是 ITER-RETGEN，它迭代检索增强生成和生成增强检索。检索增强生成根据所有检索到的知识输出对任务输入的响应。在每次迭代中，ITER-RETGEN 利用前一次迭代的模型输出作为特定上下文，以帮助检索更相关的知识。循环的终止由预定义的迭代次数确定。

1.1.4.6 递归检索（Recursive Retrieval）结构

递归检索的特点是与迭代检索相比，它明显依赖于前一步骤，并且不断加深检索。通常，递归检索会有一个终止机制作为退出条件。在 RAG 系统中，递归检索通常涉及查询转换，依赖于每次检索的新重写查询。

典型的递归检索实现，例如 ToC，涉及递归执行 RAC（递归增强澄清）来逐步将子节点插入到澄清树中，从最初的模糊问题（AQ）开始。在每个扩展步骤中，根据当前查询进行段落重新排序，生成一个明确的问题（DQ）。树的探索在达到最大数量的有效节点或最大深度时结束。一旦构建了澄清树，ToC 会收集所有有效的节点，并生成一个全面的长文本答案来回答 AQ。

ToC 参考资料：https://aclanthology.org/2023.emnlp-main.63.pdf

1.1.4.7 自适应检索（Adaptive Retrieval）结构

随着 RAG 的发展，已经逐渐从被动检索转向了自适应检索的出现，也被称为主动

检索，这在一定程度上归功于 LLM 的强大能力。这与 LLM 代理有着共同的核心概念。

RAG 系统可以主动确定检索的时机，并决定何时结束整个过程并生成最终结果。根据判断标准，这可以进一步分为基于提示和基于微调的方法。

基于提示的方法。该方法涉及使用提示词工程来控制流程，以指导 LLM。一个典型的实施示例是 FLARE。其核心概念是语言模型只在缺乏必要知识时进行检索，以避免在增强型语言模型中进行不必要或不适当的检索。FLARE 迭代生成下一个临时句子，并检查低概率标记的存在。如果找到，系统会检索相关文档并重新生成句子。

基于微调的方法。基于微调的方法涉及对 LLM 进行微调，以生成特殊标记，从而触发检索或生成。这个概念可以追溯到 Toolformer，其中生成特定内容有助于调用工具。在 RAG 系统中，这种方法用于控制检索和生成步骤。

基于微调的方法的一个典型例子是 Self-RAG。具体步骤包括：

1. 给定一个输入提示和前一代结果，首先预测特殊标记“检索”是否有助于通过段落检索来增强持续生成。
2. 如果需要检索，模型会生成：一个评估检索到的段落相关性的批评标记，下一个回应片段，以及一个评估回应片段中的信息是否得到段落支持的批评标记。
3. 最后，一个评价令牌评估响应的整体效用，并选择最佳结果作为最终输出。

Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection：https://arxiv.org/abs/2310.11511

Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity：https://arxiv.org/abs/2403.14403