ADL 147

Report 1. 知识检索增强：范式与关键技术

同济大学 王昊奋

Retrieval-Augmented Generation for Large Language Models: A Survey [arXiv24.3.27]

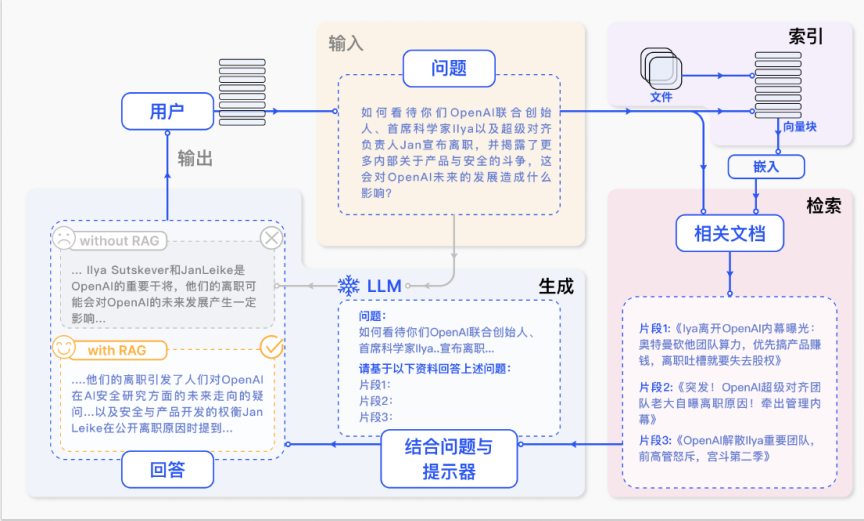
01 RAG基本概述

RAG提出背景

LLM缺陷：幻觉（一本正经的胡说八道）、信息过时、参数化知识效率低、缺乏专业领域的深度知识、推理能力弱

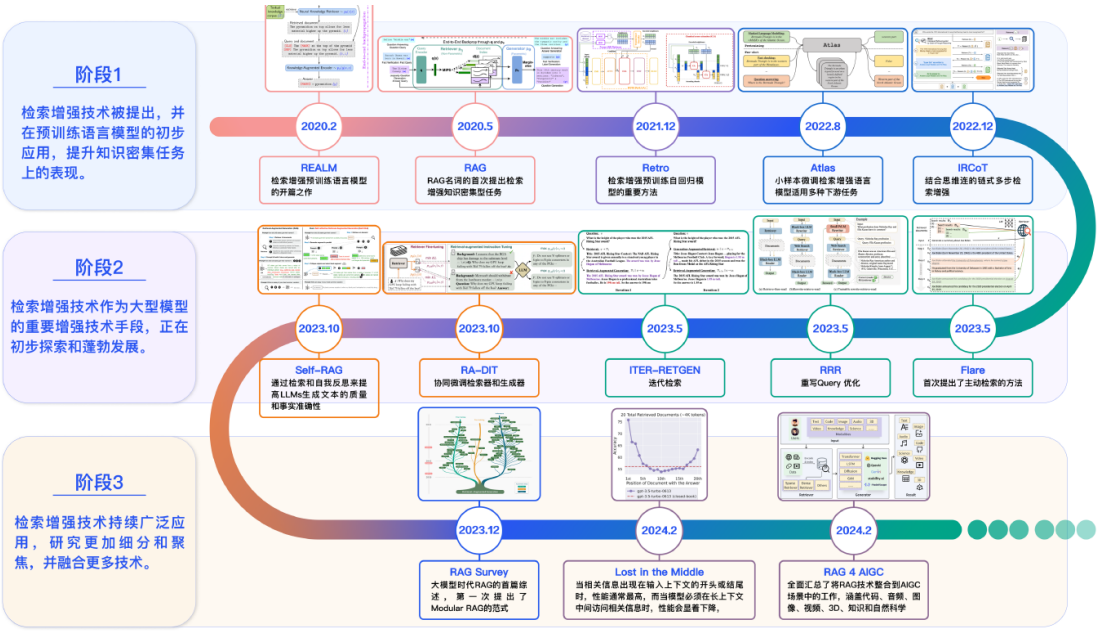
实际应用的需求：领域精准问答、数据频繁更新、生成内容可解释可溯源、成本可控、数据隐私保护

- LLM 在回答问题或生成文本时，先会从大量文档中检索出相关的信息，然后基于这些信息来生成回答。
- RAG 方法使得不必为每一个特定的任务重新训练整个大模型，只需要外挂知识库。
- RAG 模型尤其适合知识密集型的任务。



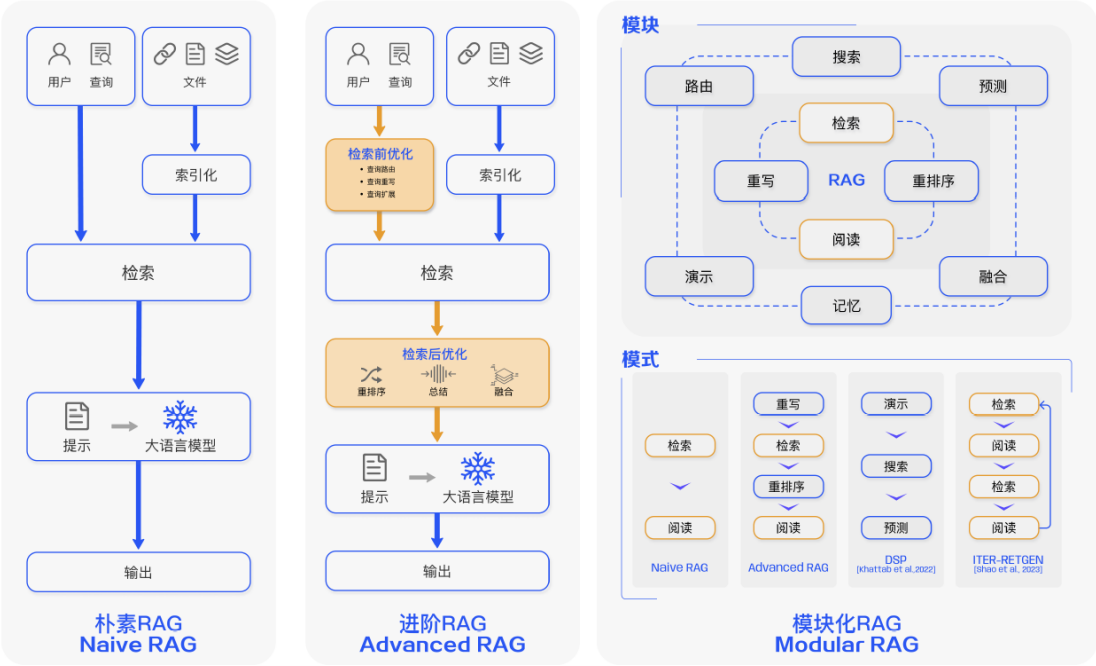
RAG 的主要流程

02 RAG的主要范式与发展历程



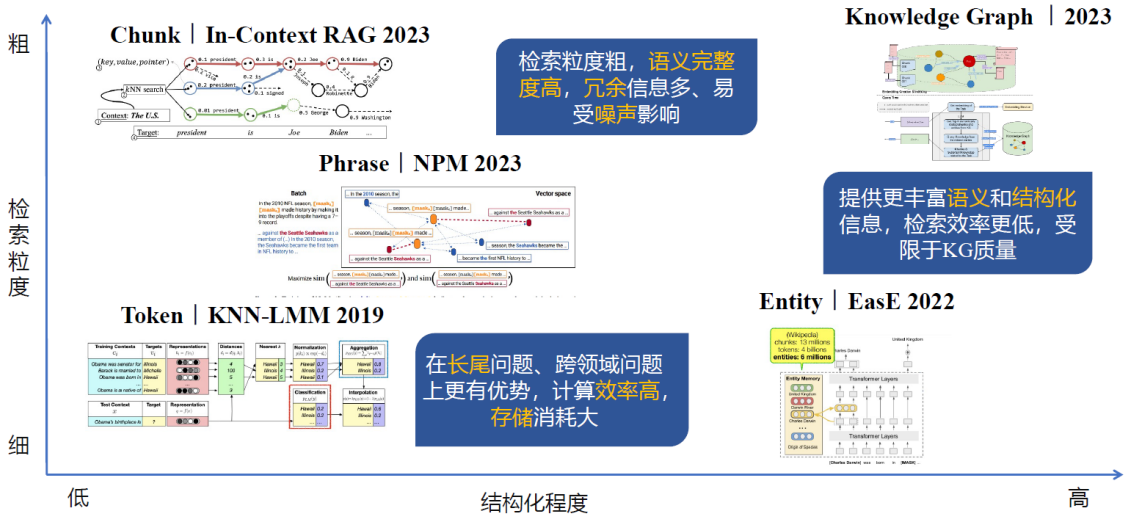
RAG的三种典型范式：

- Naive RAG：
  - 构建数据索引：文档分块、生成embedding、存储到向量数据库中
  - 检索：向量相似度度量，得到k个文档
  - 原始的query与检索得到的文本组合输入到LLM，得到回答
- Advanced RAG
  - 索引优化：滑动窗口、细粒度分割、元数据
  - 前检索模块：检索路由、摘要、问题重写、置信度判断
  - 后检索模块：重排序、检索内容过滤
- 模块化RAG



RAG三大问题：

- 检索什么？词元、词组、句子、段落、实体、知识图谱



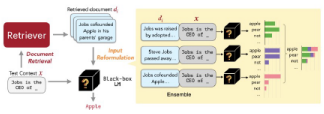
- 什么时候检索？单次检索、每个token、每N个token、自适应检索

效率高，检索到的文档相关度低

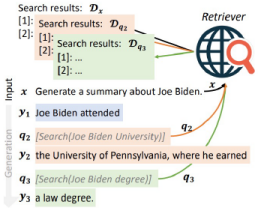
平衡效率和信息的矛盾
可能非最优解

检索到的信息量大，但效率低，冗余信息多

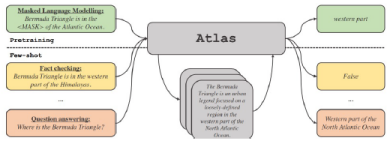
Once | Replug 2023



Adaptive | Flare 2023



Every N Tokens | Atlas 2023



在推理中仅进行一次检索

自适应地进行检索

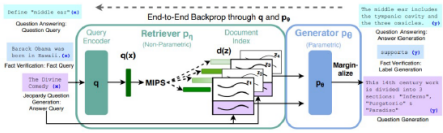
每生成N个Tokens去检索一次

低 检索频率 高

- 怎么使用检索的结果？输入/数据层、模型/中间层、输出/预测层

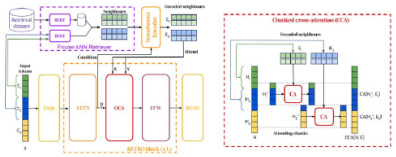
  - 在推理过程中，集成检索到的信息到生成模型的不同层级中

集成检索位置



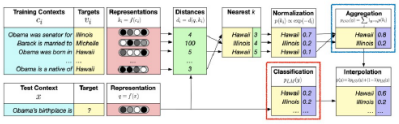
输入/数据层

使用简单，但无法支持检索更多的知识块，且优化空间有限



模型/中间层

支持输入更多的知识块检索，但引入额外的复杂度，且必须训练



输出/预测层

保证输出结果与检索内容高度相关但效率低

03 模块化RAG范式与关键技术

▶ 模块化RAG体系

三层架构

Module Type

6 大模块类型：
RAG的核心流程

Module

14个模块：
流程中的具体功能

Operator

40+算子：
特定功能的具体实现

1. Indexing 索引

面临挑战：文档块不完整的语义信息、块相似度计算不准确、参考轨迹不明确

解决方案：分块索引优化、结构化语料

分块索引优化：

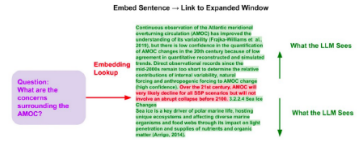
分块策略

Small-2-Big
在句子级别嵌入文本，然后在LLM生成过程中扩大窗口

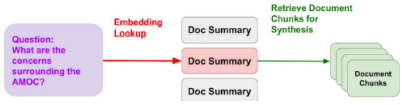
滑动窗口
滑动Chunk覆盖全文，避免语义割裂

摘要
通过摘要嵌入更大的文档。通过摘要检索文档，再从文档中检索文本块

Small-2-Big



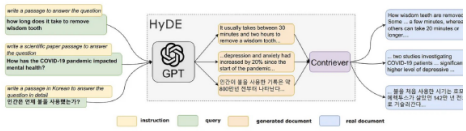
摘要



添加元数据

示例 | 页码 | 时间 | 类型 | 文档标题

伪元数据

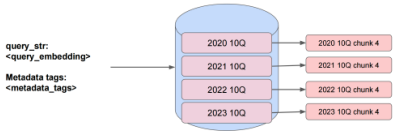


元数据筛选/扩充

伪元数据生成
通过为传入的查询生成一个假设性的文档来增强检索，并生成该文本块可以回答的问题

元数据过滤器
对文档进行分离和标记。查询期间，除了语义查询之外，并推断元数据过滤器

元数据过滤

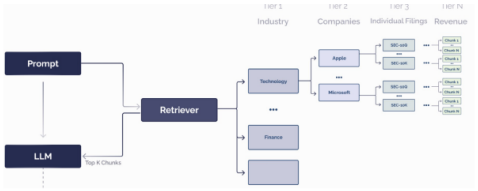


LLamaIndex [Liu et al., 2023]

结构化语料：

层级结构

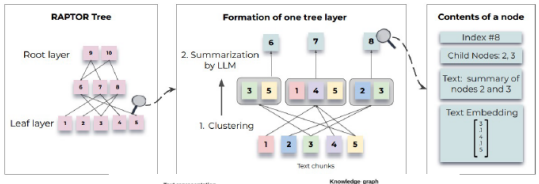
- 文档原始结构、语义结构分割
- 检索摘要->再检索具体的节点
- 图、表作为独立检索对象



Arcus
[Arcus Team, 2023]

树结构索引

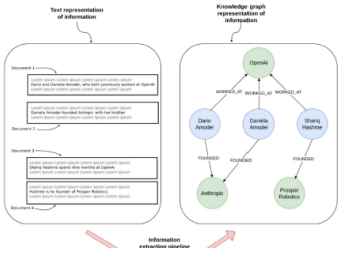
- 语义相似度聚类子节点
- 生成摘要作为父节点
- 递归构树结构索引



RAPTOR
[Sarthi et al., 2024]

知识图谱索引

- 文本块 ->三元组实例 ->图结构
- 构建文档块之间的语义关系
- 增强上下文理解和可解释性



Neo4j [Bratanic., 2024]

2. Pre-Retrieval 检索前处理

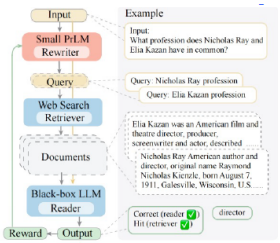
面临挑战：措辞不当的查询、语言自身的复杂性和歧义性

解决方案：查询扩展、查询转换、查询路由

查询转换：

查询重写 Query Rewrite

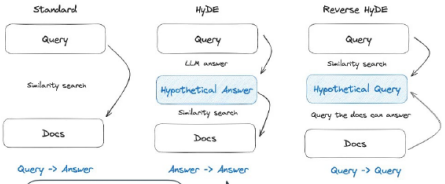
- 原始查询并不总是检索的最佳选择
- LLM / 专用小模型重写查询



Rewrite-Retrieve-Read
[Ma et al., 2023]

假设回答HyDE

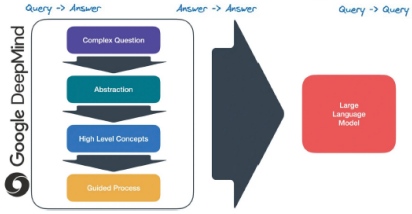
- 构建假设回答代替原始查询去检索
- 或为Chunk生成假设问题作为查询依据



HyDE
[Gao et al., 2022]

后退提示 Step-back Prompting

- 构建假设回答代替原始查询去检索
- 或为Chunk生成假设问题作为查询依据



Take a Step Back
[Zheng et al., 2023]

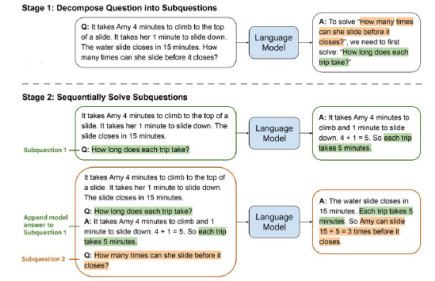
查询拓展：

多查询 Multi-Query

- 提示工程通过 LLM 扩展查询
- 根据预设的模板选择相似的查询
- 为原始查询分配更高权重避免意图稀释

子查询 Sub-Query

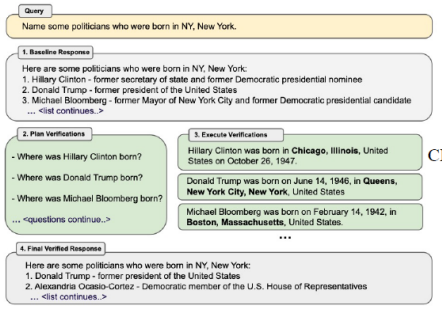
- 复杂的问题分解成一系列更简单的子问题
- 针对各个子问题分别检索增强生成
- 得到的中间结果与原问题合并



LEAST-TO-MOST PROMPTING
[Zhou et.al., 2023]

验证链 CoVe

- 通过LLM生成一组要问的验证问题
- 通过回答这些问题并检查是否一致执行



CHAIN-OF-VERIFICATION (CoVe)
[Dhuliawala et.al., 2023]

查询路由：

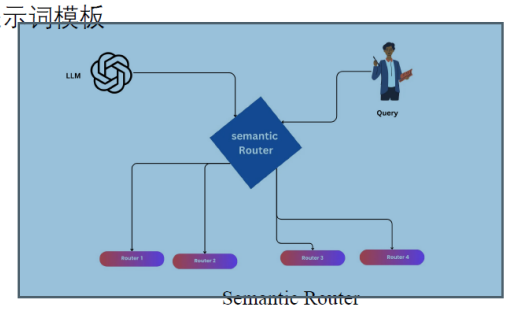
根据不同的场景，路由到不同的RAG流程、组件或提示词模板

元数据路由器/过滤器 Metadata Router/ Filter

提取查询中关键字（实体），根据关键字和元数据进行筛选

语义路由器 Semantic Router

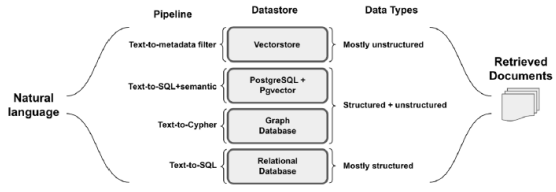
根据用户自然语言的意图选择RAG的流程
将基于语义和元数据的方法结合起来



Semantic Router
[Aurelio AI]

Text-to-Cypher / Text-to-SQL

将用户的自然语言查询转换为另一种查询语言
以便查询其他结构化数据源



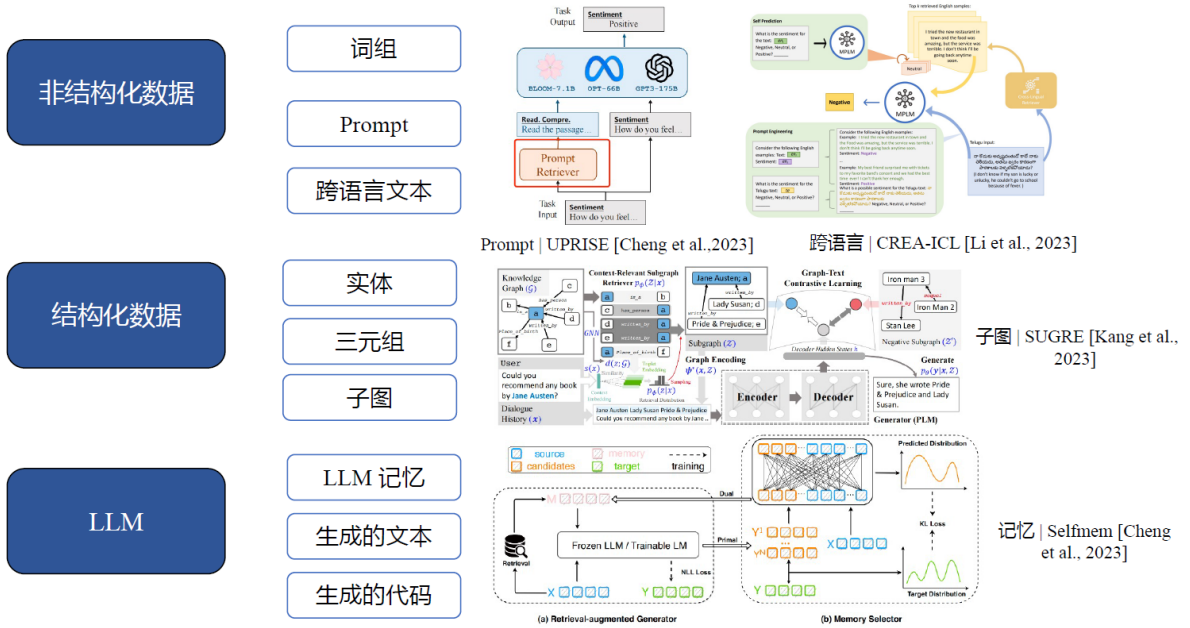
Query Construction
[Langchain]

3. Retrieval 检索

面临挑战：检索效率、嵌入表示质量、任务 数据和模型的一致性

解决方案：检索元选择、检索器选择、检索微调

检索元选择：



检索器选择：

稀疏检索器Sparse Retriever

- 基于统计方法
- BM25, TF-IDF

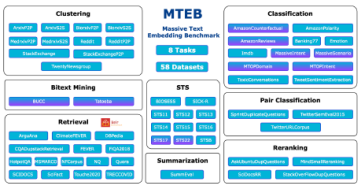
密集检索器Dense Retreiver

- 基于语言模型
- Encoder-Decoder架构，例如，BERT系列
- 多任务微调，例如BGE
- 基于Decoder LLMs 微调，例如GTE-Qwen

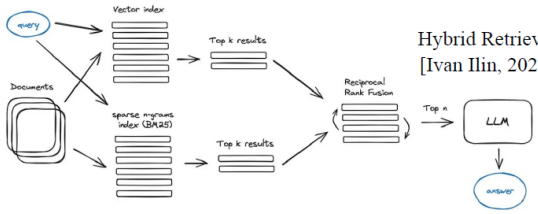
混合/混合检索 Mix/hybrid Retrieval

- 同时使用两种检索方法，特征互补
- 稀疏检索作为密集检索的初始搜索结果

Embedding 供应商



MTEB
[Muennighoff et.al., 2022]



Hybrid Retrieval
[Ivan Ilin, 2023]

检索器微调：

有监督微调SFT

- 特定领域 / 通用领域（混合）的监督数据

语言模型监督微调
LM-supervised Retriever

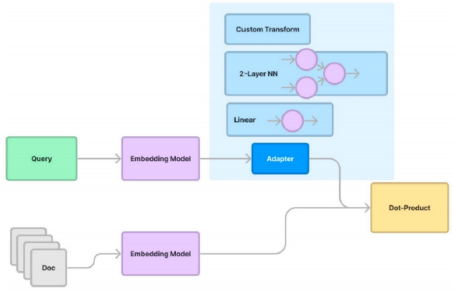
- 利用生成的结果作为监督信号，在 RAG 过程中对嵌入模型进行微调。

强化学习微调

- 受RLHF的启发，通过强化学习来强化检索器

适配器 Adapter

- 微调检索器的成本较高 / 无法微调的闭源模型
- 特定任务适配器Task Specific
- 通用适配器Task Agnostic



Embedding Adapter
[LlamaIndex]

4. Post-Retrieval 后检索

面临挑战：噪音/反事实文档、上下文窗口影响

解决方案：重排序rerank、上下文压缩筛选、检索器微调

重排序：

在不改变其内容或长度的情况下对检索到的文档块进行重新排序，增强更重要文档块的可见性

基于规则的重新排名 Rule-based Rerank

- Diversity
- Relevance
- MRR

基于模型的重新排名 Model-based Rerank

- Encoder-Decoder 模型，e.g. SpanBERT
- 专用排序模型，e.g. Cohere Rerank，BGE-reranker
- 通用LLM, e.g. GPT-4

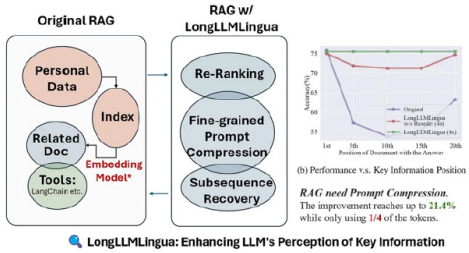
Cohere Rerank（Rule-based / Model-based)

压缩与筛选：

RAG一个常见的误解是认为检索尽可能多的相关文档连接起来形成一个冗长的提示是有益的。然而，过多的上下文可能会引入更多的噪声，削弱LLM对关键信息的感知，

(Long)LLMLingua : 删除冗余Token

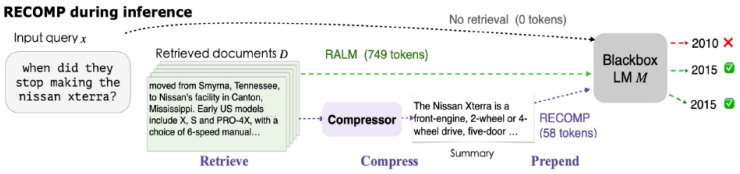
- 经过对齐和训练的小型语言模型从提示中检测和删除不重要的Token



LongLLMLingua: Enhancing LLM's Perception of Key Information

Recomp:压缩上下文

- 从检索到的文档中选择相关句子的抽取式压缩器。
- 合并来自多个文档的信息产生简洁摘要的摘要式压缩器。



Selective Context 上下文选择

- 识别并去除输入上下文中的冗余内容。
- 类似“停用词移除”策略。
- 基于LM自信息来评估Token的信息量，保留具有更高自信息量的内容

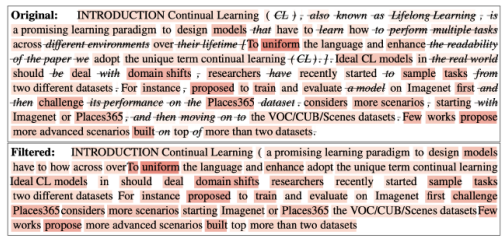
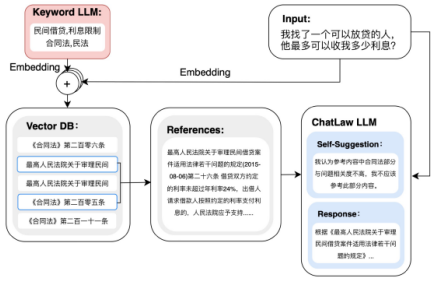


Figure 2: A visualisation of selective context. Darker colour indicates larger value of self-information.

LLM-Critique 语言模型判断

- 直接 LLM 在生成最终答案之前评估检索到的内容。通过 LLM 评论过滤掉相关性较差的文档。
- 例如，在Chatlaw中，LLM被要求对所引用的法律条款进行自我建议，以评估其相关性。



5. Generation 生成

面临挑战：LLM的选型、缺乏领域知识、复杂问题推理能力有限、LLM的幻觉

解决方案：生成器选型、生成器微调、事实校验与知识编辑

模型选型：

闭源/ 开源 模型?

- 模型性能。闭源模型通常由更好的性能
- 数据隐私。内部数据会传输给供应商
- 模型调整。开源模型可以进一步训练微调

开源模型（绿线）正在逼近闭源模型的效果（红线）

Cloud API / 本地部署 ?

- 计算资源。本地需要更多的计算资源
- 并发与时延。Cloud API拥有更大的弹性
- 数据保护。本地部署可以更有效保护隐私

通过云API使用LLM正在成为更具性价比的选择

生成器微调：

有监督指令微调

- 领域 / 通用指令微调补充领域信息
- 适应特定的输入结构（文本对、图结构等）

强化学习

- 强化学习对齐模型在使用检索文档上的偏好

模型蒸馏

- 使用更强大的模型作为教师模型微调生成器

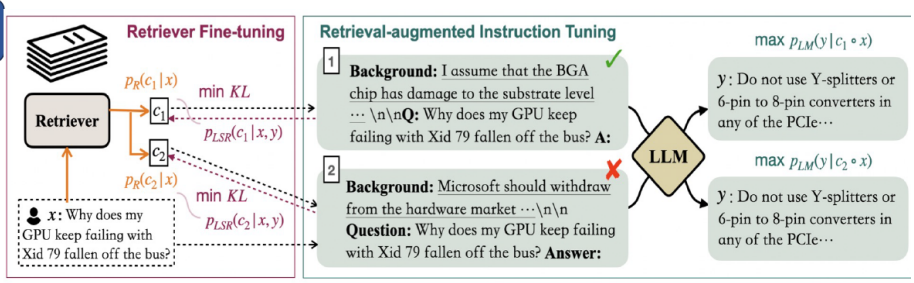
生成器与检索器协同微调

- **R-FT**

最小化检索器分布与LLM偏好之间的KL散度

- **LM-FT**

最大化给定检索增强指令情况下正确答案的可能性



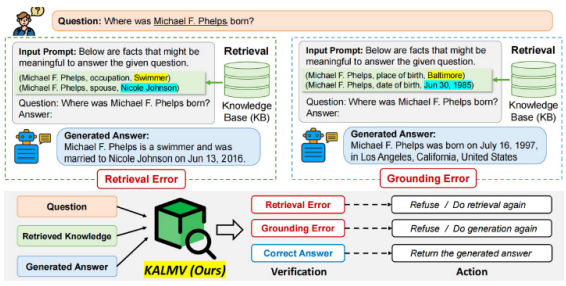
RA-DIT [Lin et al., 2023]

事实校验：

经过检索增强并不能确保无幻觉生成，尤其是检索到噪声或冲突事实时，生成后再校验减少幻觉。

模型验证

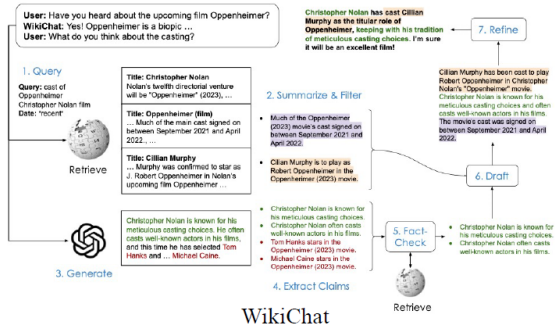
- 用准确知识（例如KG）训练的小模型用于大模型的输出校验



KALMV
[Baek et. al., 2023]

事实数据交叉校验

- 用经过校验的知识（例如Wikipedia）验证答案的时效性和准确性



WikiChat
[Semnani et. al., 2023]

6. Orchestration 编排

面临挑战：传统的链式且一次性的检索-生成流程不足以解决复杂推理或设计大量知识的任务

解决方案：检索流程调度、知识引导检索流程、检索流程聚合

检索流程调度：

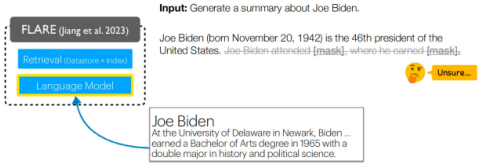
评估RAG过程中的临界点，判断是否需要检索外部文档库、答案的满意度以及是否需要进一步探索。常用于递归、迭代和自适应检索。

基于规则

- 生成的答案会被评分，根据分数是否满足预定义的阈值做出继续或停止的决定。
- 常见的阈值包括Token的置信度水平

基于提示

- 通过提示工程对当前答案进行反思
- 由LLM动态的判断检索的范围和生成终止



FLARE [Jiang et al., 2023]

特殊Token

- 生成过程中根据特殊的Token判断下一步的行为
- 通常需要微调以满足特定的输出格式

Type	Input	Output	Definitions
Retrieve	x / x, y	{yes, no, continue}	Decides when to retrieve with $\mathcal{R}$
ISREL	x, d	{**relevant**, irrelevant}	d provides useful information to solve x.
ISSUP	x, d, y	{**fully supported**, partially supported, no support}	All of the verification-worthy statement in y is supported by d.
ISUSE	x, y	{5, 4, 3, 2, 1}	y is a useful response to x.

Algorithm 1 SELF-RAG Inference

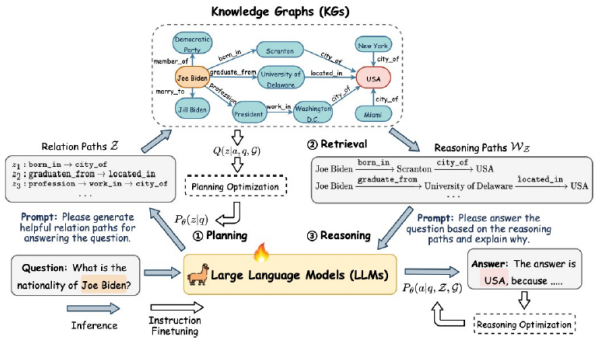
Require: Generator LM $\mathcal{M}$, Retriever $\mathcal{R}$, Large-scale passage collections $\{d_1, \ldots, d_N\}$
Input: input prompt x and preceding generation $y_{<t}$, **Output:** next output segment y_t
$\mathcal{M}$ predicts Retrieve given $(x, y_{<t})$
if Retrieve == Yes **then**
  Retrieve relevant text passages D using $\mathcal{R}$ given (x, y_{t-1}) ▷ Retrieve
  $\mathcal{M}$ predicts ISREL given x, d and y_t given $x, d, y_{<t}$ for each $d \in$ D ▷ Generate
  $\mathcal{M}$ predicts ISSUP and ISUSE given x, y_t, d for each $d \in$ D ▷ Critique
  Rank y_t based on ISREL, ISSUP, ISUSE ▷ Detailed in Section 3.3
else if Retrieve == No **then**
  $\mathcal{M}_{gen}$ predicts y_t given x ▷ Generate
  $\mathcal{M}_{gen}$ predicts ISUSE given x, y_t ▷ Critique

Self-RAG [Asai et al., 2023]

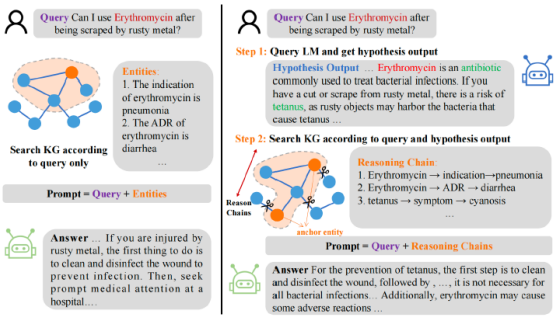
知识引导：

通过知识库来指导RAG的检索与生成流程，增强RAG流程的可追溯性，提高生成的可靠性

1. 计划 - 检索 - 推理
2. 生成以KG为依据的计划执行路径
3. 根据计划从KG和语料中检索相关内容
4. 生成具有推理路径和可解释性的回答



RoG [Luo et al., 2023]



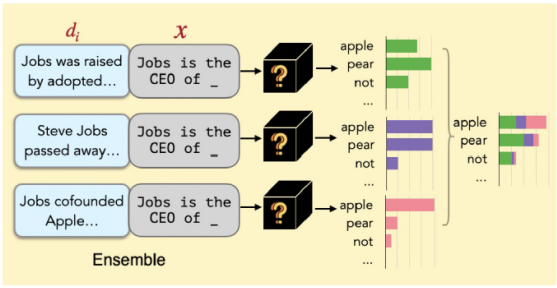
HyKGE [Jiang et al., 2023]

流程聚合：

当RAG不再是单一的流水线，多个分支扩展检索范围和多样性，融合模块对多个答案进行融合。

基于概率聚合

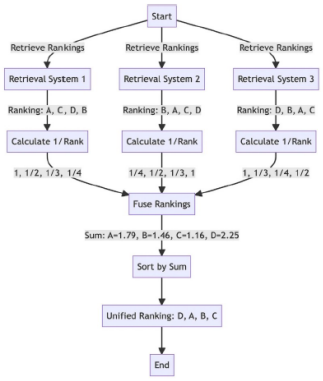
- 融合方法从多个分支生成的不同token的加权值，从而对最终输出进行综合选择。
- 主要采用加权平均法。



RePlug [Shi et. al., 2023]

Reciprocal Rank Fusion (RRF)

- 将多个搜索结果列表的排名结合起来生成单一统一排名的技术。



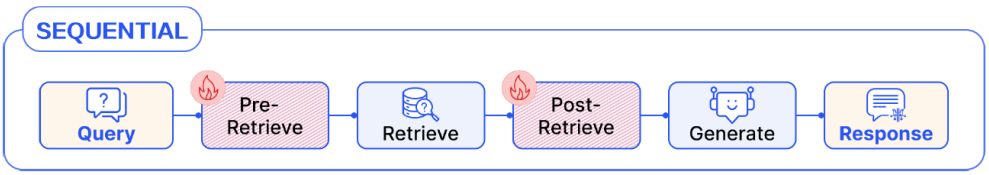
RAG Fusion [Raudaschl, 2023]

7. RAG FLOW

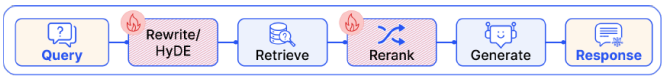
Flow Pattern	特点		适用场景
Sequential	简单有效	复杂任务处理能力弱	难度不大的场景 适合作为**快速上手**
Conditional	多路由配置，适合多任务场景	每个路由分支仍是Sequential结构，复杂任务能力弱	数据**结构丰富**，同时包括文本、表格、图结构等
Braching	拆解复杂查询，分别检索生成后聚合，复杂任务处理能力提升	分支后需要聚合，各项开销提高	查询较复杂，**易于分解**成多个子问题，容忍一定的时延
Loop	多次RAG，具有较高的自主性和灵活性。复杂任务能力较高	可控性下降，时延和开销较大	任务**困难**，对**时延要求不高**，允许模型有一定灵活性的**非严肃任务**场景

线性RAG FLOW

Sequential Flow Pattern 各个模块线性的组织，经典的Naive RAG和基础的Advanced RAG均为线性结构。

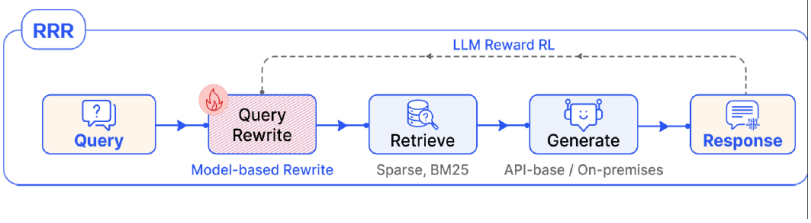


常在检索前增加Query Rewrite，在检索后增加Rerank算子提升整体效果。
例如QAnything



Rewrite-Retrieve-Read

- Query Rewrite一个小型的可训练LM
- 通过最终LLM的输出结果作为奖励
- 在强化学习的背景下，重写器优化被形式化为一个马尔科夫决策过程

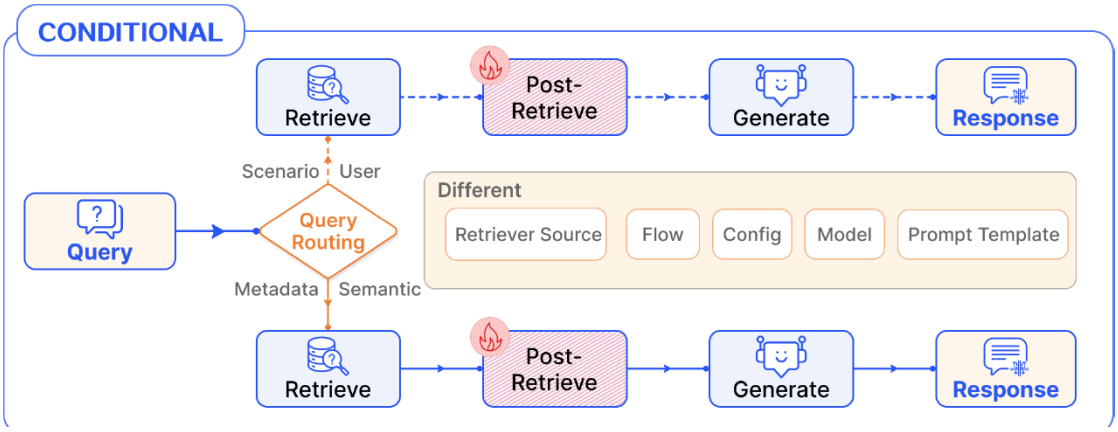


条件RAG FLOW

Conditional Flow Pattern

根据不同的条件选择不同的RAG路线。通常由一个Routing模块进行路由。

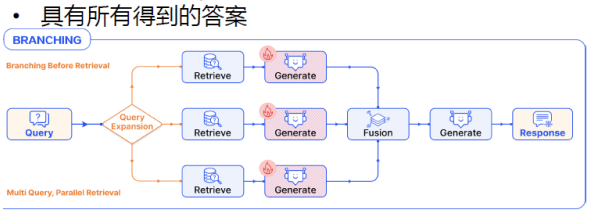
- 判断依据包括：语义、任务类型、元数据、用户权限。
- 不同路由分支在检索源、检索流程、配置信息、模型选择和Prompt Template上进行差异化。
- 严肃场景下的任务（例如 医疗、法律等）与娱乐问题，对大模型幻觉的容忍度是不同的。



分支RAG FLOW

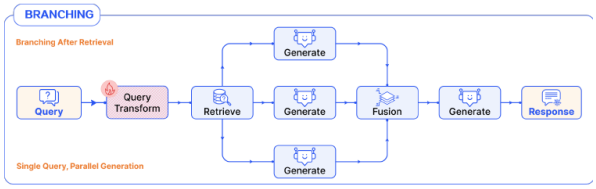
检索前分支

- 对原始Query进行扩展，得到多个Sub-Query。
- 分别对Sub-Query进行检索、生成
- 具有所有得到的答案



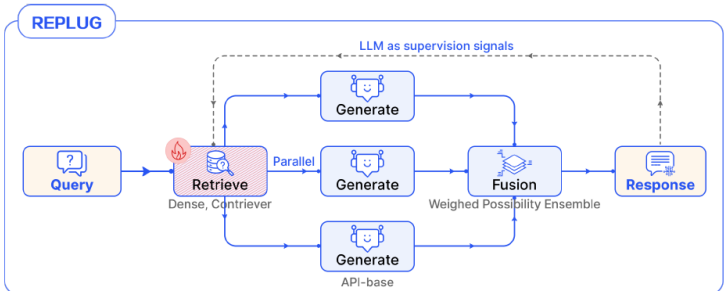
检索后分支

- 保持原来的Query，检索到多个文档块后，
- 并行使用原始Query和每一个文档块进行生成
- 对多个生成结果进行聚合



REPLUG

- 典型的检索后分支的分结构
- 根据每一个分支预测token的概率
- 通过Weighted possibility Ensemble 将不同的分支聚合。
- 并通过最后生成结果作作为反馈微调检索器Contriever。



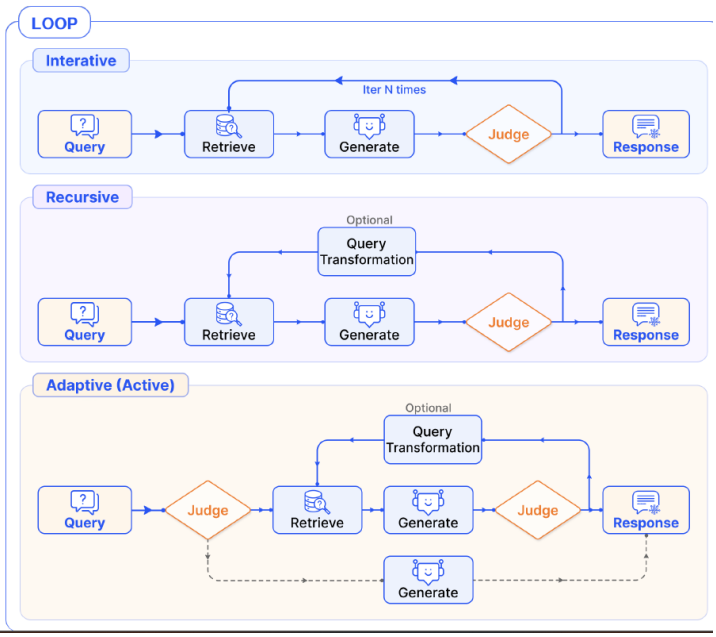
环状RAG FLOW

Loop Flow Pattern

- 面对复杂问题，一次的检索生成效果并不理想，通过多次检索增强
- 具有循环结构的RAG Flow，是Modular RAG的重要特点。
- 检索和推理步骤相互影响的。通常包括Judge模块，用于控制流程。

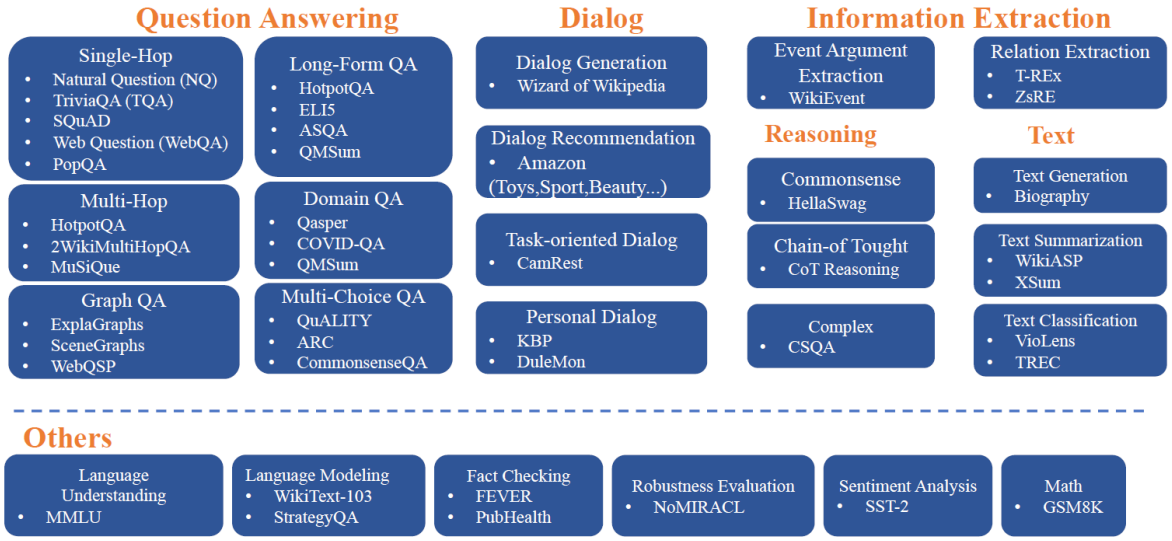
具体又可以分成:

- 迭代检索
- 递归检索
- 自适应（主动）检索



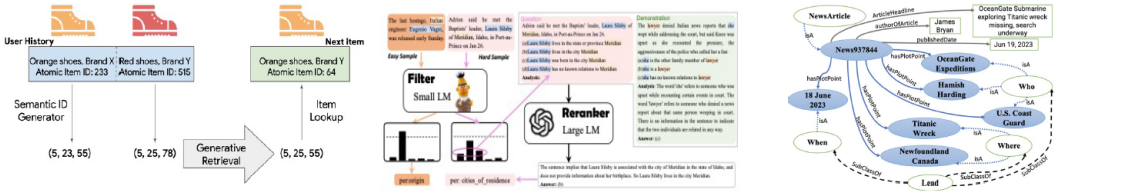
04 RAG的下游任务与评估

常用数据集:

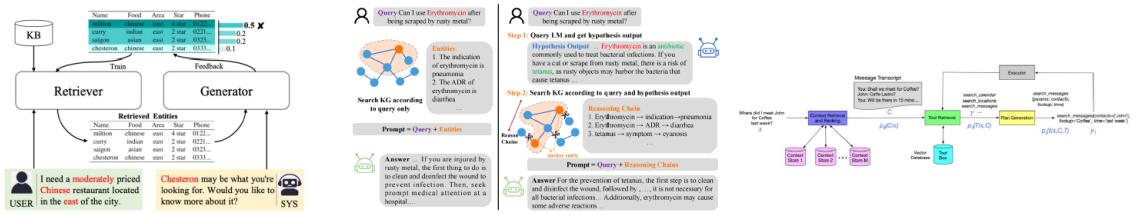


下游任务：

进一步扩展RAG的下游任务、完善生态建设



推荐系统 | TIGER [Rajput et al.,2023] 信息抽取 | Filter- Rerank [Ma et al.,2023] 报告生成 | FABULA [Ranade et al.,2023]



对话系统 | ToD [Shi et al.,2023] 医疗问答 | HyKGE [Jiang et al.,2023] 端侧应用 | CT-RAG [Anantha et al., 2023]

评估方法：

目前评估依赖于WiKipedia Dump数据集（40G），进行评估成本较高，并且很多评估数据集的语料库都用于模型训练存在泄露问题；外面需要更低成本的更公正的面向RAG的评测体系。

05 RAG的工具栈与行业应用

个人知识助手

Quivr: Your second brain

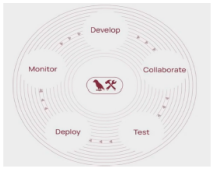
WhyHowAI

问答系统 Linkdin智能客服

软件工程 CodeGeeX

技术栈与工具

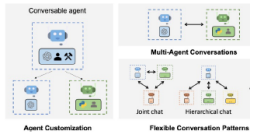
名称	优点	缺点
LangChain	模块化，功能全面	行为不一致并且隐藏细节 API复杂，灵活度低
LlamaIndex	专注知识检索	需组合使用，定制化程度低
FlowiseAI	上手简单，流程可视化	功能单一，不支持复杂场景
AutoGen	适配多智能体的场景	效率低，需要多轮对话



LangChain


LlamaIndex


FlowiseAI



AutoGen

- 用途**定制化**，满足蛮多样的需求
- 使用**简易化**，进一步降低上手门槛
- 功能**专业化**，逐渐面向生产环境

AnythingLLM

LangChain-Chatchat

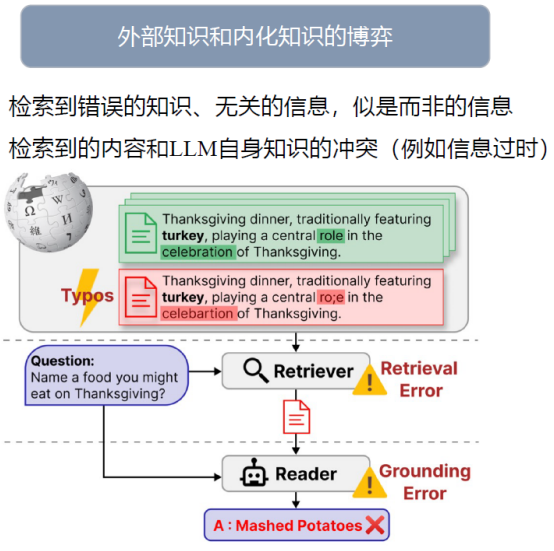
RAGFlow

QAnything

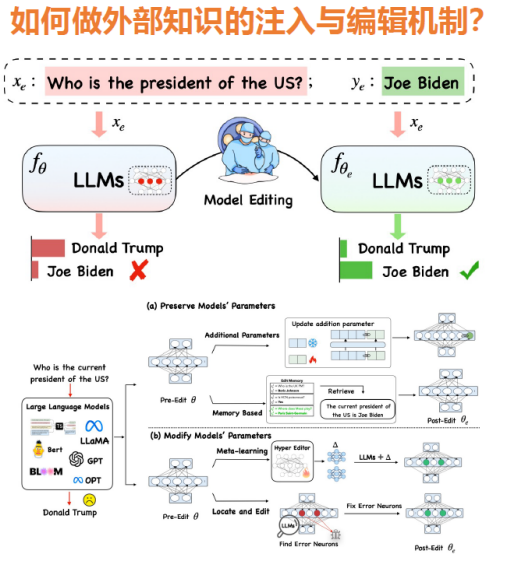
06 RAG的挑战与机遇

理论侧：

外部知识和内化知识的博弈、RAG记忆与遗忘机制



GARAG | [Cho et.al.,2024]



Editing Large Language Models: Problems, Methods, and Opportunities

数据侧：

多模态数据检索与理解、长文本的高效切分 向量化与检索、大规模知识库的管理、异构数据的整合与存储、评测集与指标

大规模知识库的管理

完成RAGDemo是容易的，构建企业级的RAG应用是困难的：

- 在大规模知识库的设计与构建
- 大规模数据上的检索与推理效率
- 知识的淘汰与更新

多模态数据检索与理解

代码、表格、图像、视频都拥有丰富的知识
如何在RAG的范式下将这些数据关联起来，共同作为LLM的上下文

异构数据的整合与存储

- 不同格式数据中不同的向量整合问题
- 超长维度的Embedding存储（例如7111维度）
- Embedding模型更新后，如何与旧的知识库整合
- （例如之前使用了Ada002，现在改用text-embedding3）

长文本的高效切分、向量化与检索

当用户上传了一份超长的文件，例如小说、著作、博士论文、企业年报
如何在可接受的时间内响应用户的查询？

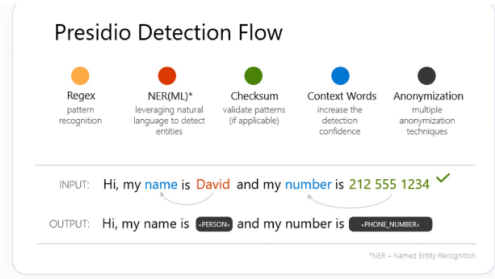
技术侧：

幻觉控制、隐私保护、自主控制的RAG流程、流程可追溯 结果可解释、查询与知识的语义距离

隐私保护

语料库中可能存在着大量的隐私信息

- 希望既可以被用来补充上下文，同时又不希望他们直接泄露出去
- 不同用户使用RAG系统的权限控制
- 如何利用无法传输到云端的隐私数据



自主控制的RAG系统

- LLM越来越多的参与了RAG的流程，例如生成检索语料、决策RAG流程

我们距离自主控制的RAG系统还有多远？

- LLM学会人类的学习方法：从问题中抽象、联想，模仿，举一反三
- 自主判断RAG流程：
  - 是否需要去检索
  - 何时停止检索
  - 生成的答案的是否可靠
  - 何时结束生成

幻觉控制

- 尽管目前的RAG方法有效降低了LLM的幻觉,但仍无法确保零幻觉。
- 在某一些严肃场景下，需要LLM接近100%的零幻觉。如何做生成后的校验，实现零幻觉

法律场景（LexiLaw）

医疗场景（Ming明医）

流程可追溯,结果可解释

- 有时候重要的不是回答什么，而是如何解释你的回答
- 当LLM给出错误、缺乏逻辑的回答时，我们会想要知道为什么，以及如何修正

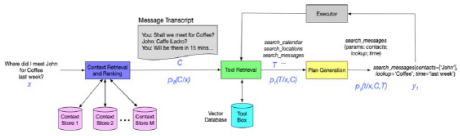
推荐任务中的可解释推荐

应用侧：

大规模工业应用、低资源任务场景

工程应用要求	低资源场景
➢ 大模型与RAG的工业应用对工程应用提出了更高的要求	➢ LLM参数受限、Embedding模型的维度受限，编码和检索速度要求高
➢ 响应时延、模型选择、检索效率、推理效率、存储成本、计算成本、性能冗余等等因素之间的权衡	➢ 在资源受限的情况下，例如端侧场景中，如何构建RAG应用

Kimi有点累了，可以晚点再问我一遍。



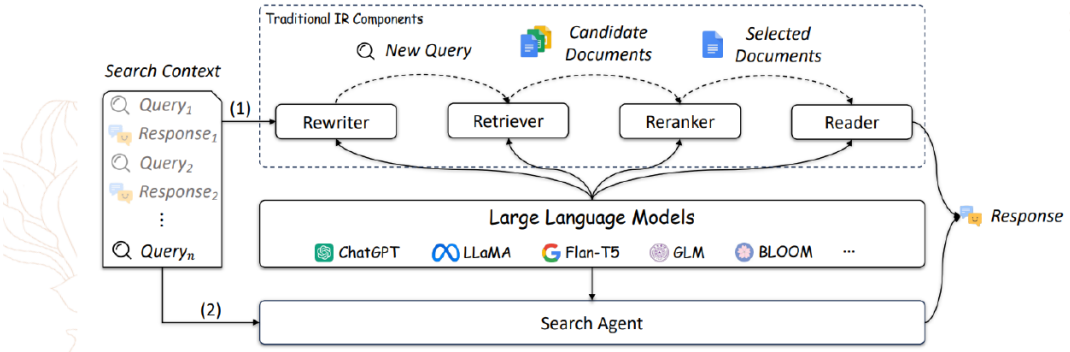
端侧应用 |Apple CT-RAG [Anantha et al., 2023]

Report 2. 大语言模型与智能信息检索技术的融合

中国人民大学高瓴人工智能学院 窦志成 朱余韬

PART 1 大模型赋能的信息检索

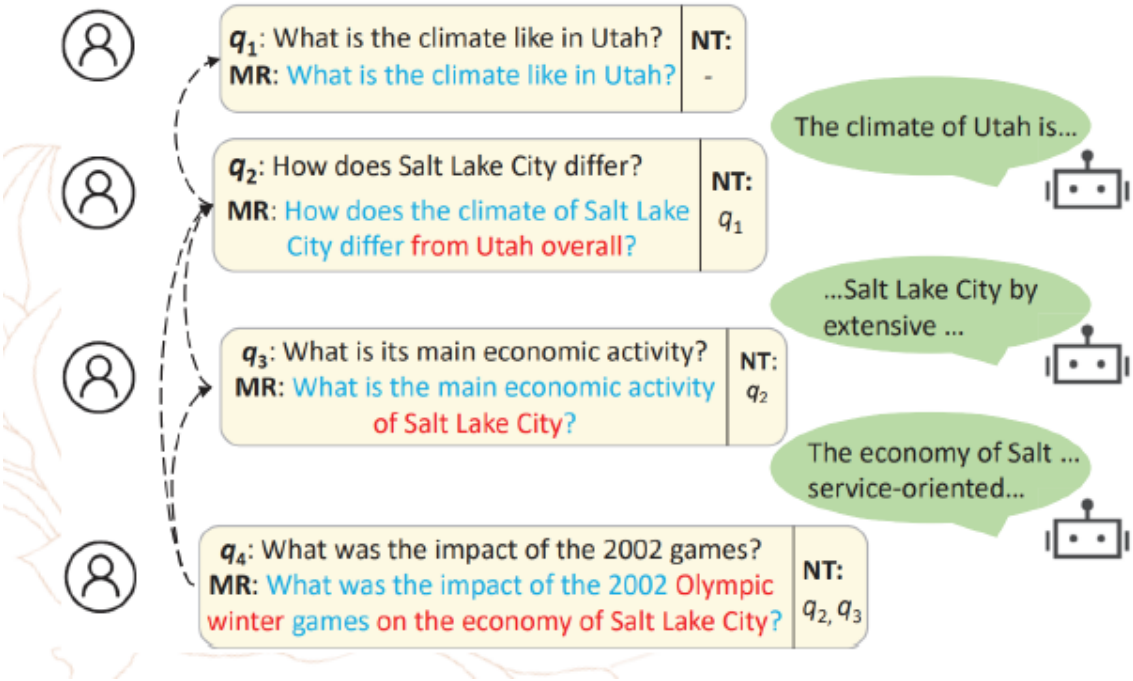
Large Language Models for Information Retrieval: A Survey arXiv23.8.14



1. Rewriter 查询改写

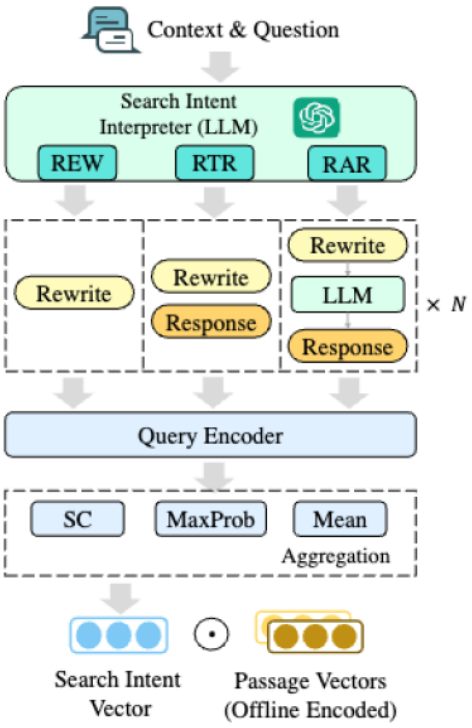
为什么要进行查询改写？

- 原查询过短或模糊，大模型可以更好的理解用户意图
- 在对话系统中，改写更为重要，继承上下文



基于LLM进行对话式查询改写

> Large Language Models Know Your Contextual Search Intent:A Prompting Framework for Conversational Search [EMNLP 2023]



Instruction	For an information-seeking dialog, please help reformulate the question into rewrite that can fully express the user's information needs without the need of context, but also generate an informative response to answer the question.
Demonstration	I will give you several example multi-turn dialogs, where each turn contains a question, a response, and a rewrite. The rewrite part begins with a sentence explaining the reason for the rewrite. **Example #1:** *Question*: What should I consider when buying a phone? *Rewrite*: This is the first turn. So the question should be rewritten as: What should I consider when buying a phone? *Response*: The design of the phone and the overall ... *Question*: Cool. Which one would you recommend? *Rewrite*: Based on Turn 1, you are inquiring about what should be considered when buying a phone. So the question should be rewritten as: Cool. Which smartphone would you recommend for me? *Response*: Just because a phone has everything… … **Example #2:** …
Input	Your Task (only questions and responses are given): **Context**: *Question*: What was the basis of the Watergate scandal? *Response*: ... *Question*: ... *Response*: ... … **Current Question**: So what happened to Nixon? **Rewrite**: So what happened to Nixon after the events of the Watergate scandal? (Now, you should give me the rewrite and an informative response of the **Current Question** based on the **Context**. The output format should always be: Rewrite: $Reason. So the question should be rewritten as: $Rewrite\nResponse: $Response. Go ahead!)
Model Output	*Rewrite*: Based on all previous turns, Nixon was badly involved in the Watergate scandal. So the question should be rewritten as: So what happened to Nixon after the events of the Watergate scandal? *Response*: With the mounting evidence and loss…

Figure 4: A general illustration of the prompt of LLM4CS. The prompt consist of three parts, i.e., *Instruction*, *Demonstration*, and *Input*. The red part is for REW prompting, the blue part is for the RTR and RAR promptings, and the orange part is for RTR prompting. The green part is for our designed chain-of-thought.

- 常见的三种提示学习方法
  - Zero-shot
  - Few-shot：提供样例
  - 思维链：要求生成答案前提供推理过程

Methods	Prompts
	Zero-shot
HyDE [92]	Please write a passage to answer the question. Question: {#Question} Passage:
LameR [99]	Give a question {#Question} and its possible answering passages: A. {#Passage 1} B. {#Passage 2} C. {#Passage 3} ... Please write a correct answering passage.
	Few-shot
Query2Doc [92]	Write a passage that answers the given query: Query: {#Query 1} Passage: {#Passage 1} ... Query: {#Query} Passage:
	Chain-of-Thought
CoT [93]	Answer the following query: {#Query} Give the rationale before answering
CoT/PRF [93]	Answer the following query based on the context: Context: {#PRF doc 1} {#PRF doc 2} {#PRF doc 3} Query: {#Query} Give the rationale before answering

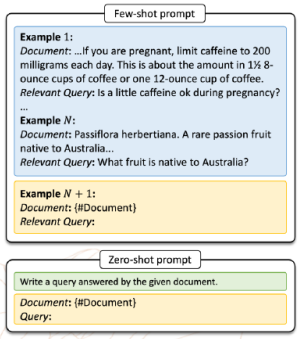
2. Retriever 检索器

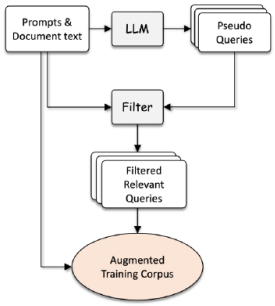
从海量文档中**高效高质**的返回相关结果.

挑战：查询模糊、文档内容多、信息复杂、标注开销大等

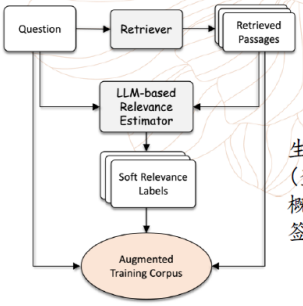
基于大模型生成检索器的训练数据

生成伪查询（构造查询-文档对）





Framework of pseudo query generation



Framework of relevance label generation

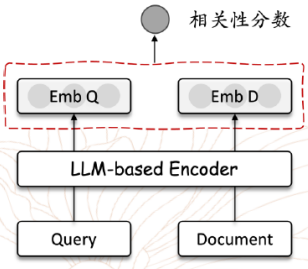
生成软标注（查询的生成概率作为标签）

Methods	# Examples	Generator	Synthetic Data	Filter Method	LLMs' tuning
InPairs [117]	3	Curie	Relevant query	Generation probability	Fixed
Ma et al. [118]	0-2	Alpaca-LLaMA & tk-Instruct	Relevant query	-	Fixed
InPairs-v2 [119]	3	GPT-J	Relevant query	Relevance score from fine-tuned monoT5-3B	Fixed
PROMPTAGATOR [120]	0-8	FLAN	Relevant query	Round-trip filtering	Fixed
TQGen [121]	0	T0	Relevant query	Generation probability	Fixed
UDAPDR [122]	0-3	GPT3 & FLAN-T5-XXL	Relevant query	Round-trip filtering	Fixed
SPTAR [123]	1-2	LLaMA-7B & Vicuna-7B	Relevant query	BM25 filtering	Soft Prompt tuning
ART [124]	0	T5-XL & T5-XXL	Soft relevance labels	-	Fixed

以大模型为基座的检索模型

Task-aware Retrieval with Instructions arXiv2211

- 稠密检索与双塔结构
  - 通用模型：使用GPT、T5-XXL等大模型初始化编码器并继续优化
  - 任务感知模型：使用提示加入任务信息

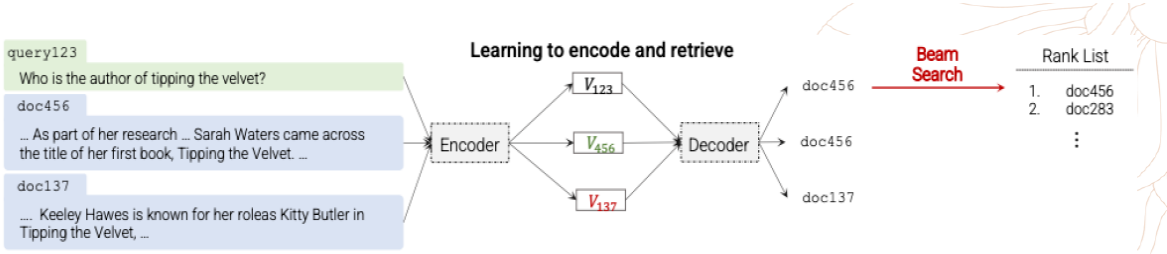


Dataset	Instruction
NQ	Retrieve a Wikipedia paragraph that answers this question.
QReCC	Find a dialogue response from dialogue history to answer the user's question.
Arguana	Retrieve a paragraph from an argument website that argues against the following argument.
SciFact	Find a sentence from a scientific paper to check if the statement is correct or not.
MultiLexSum	I want to find the one-sentence summary of this legal case.

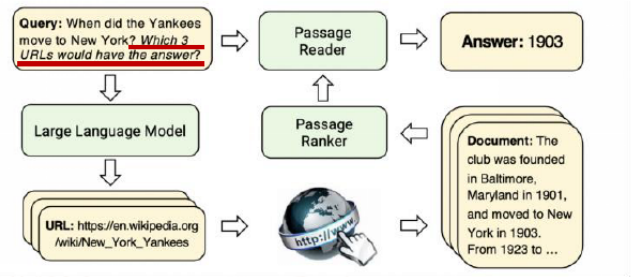
Table 1: Example instructions for Natural Questions (NQ; Kwiatkowski et al. 2019), QReCC (Anantha et al., 2021), Arguana (Wachsmuth et al., 2018), SciFact (Wadden et al., 2020) and MultiLexSum (Shen et al., 2022). Each instruction defines *intent*, *domain* and *unit*. The full list of instructions are in Appendix A.5.

大模型改善生成式检索

Transformer Memory as a Differentiable Search Index [NeurIPS 2022]



Large Language Models are Built-in Autoregressive Search Engines [ACL 2023]



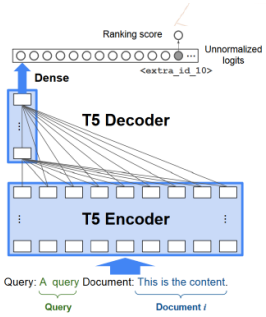
- 大模型生成网址（for Wiki）
- 用对应内容+Reader完成查询

3. Reranker 重排序器

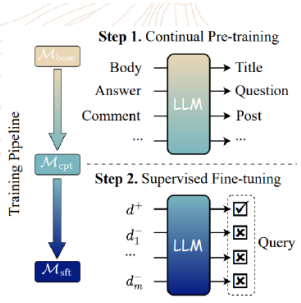
微调大模型做重排序

通常有好的性能，但训练开销大

- 不同结构
  - Encoder-only: [CLS] query [SEP] document [SEP]
  - Encoder-Decoder: RankT5
  - Decoder-only: RankLLaMA, RankingGPT
    - query: {query} document: {document} [EOS]
- 不同训练方式
  - 使用生成目标函数：拼接查询和文档，使用生成True/False的概率做优化
  - 使用排序函数：拼接查询与文档，使用特殊token生成的概率作为相关性分数，并使用排序损失函数优化



RankT5: Fine-Tuning T5 for Text Ranking with Ranking Losses, Zhuang et al., 2023

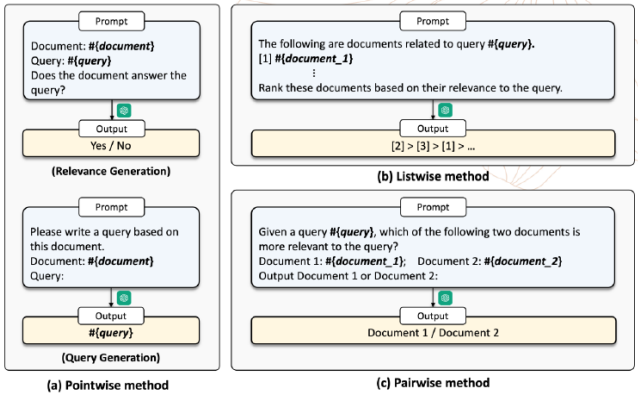


TSRankLLM: A Two-Stage Adaptation of LLMs for Text Ranking, Zhang et al., 2023

提示大模型做重排序

需要大模型能力足够强大

- 微调的成本问题 → 提示学习
  - Pointwise：判断查询和文档之间是否相关
  - Listwise：重排文档列表
  - Pairwise：比较两个文档与查询之间的相关性强弱

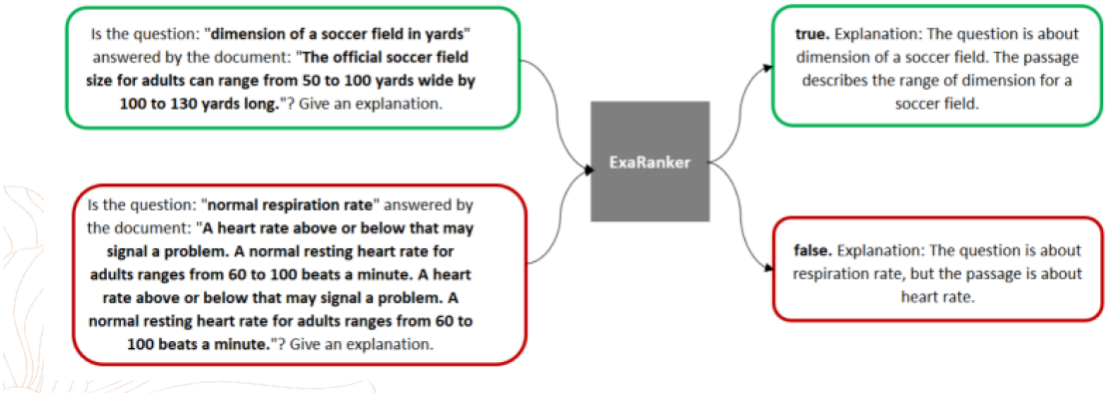


大模型生成排序数据

提升已有排序模型的有效策略

ExaRanker: Explanation-Augmented Neural Ranker [arXiv2301]

使用chatGPT生成解释作为额外的标签



4. Reader 阅读器

基于大模型对检索到的文档进行提炼总结，得到最终的答案输出

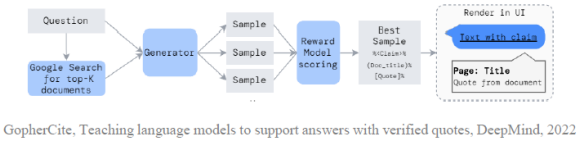
- 新型搜索引擎
  - New bing 百度AI搜索
- 商业大模型
  - Kimi chat Baichuan
- 效果仍有巨大提升空间
  - 幻想
  - 引用不相关内容
  - 编造内容
  - 错误编号

5. 搜索Agent

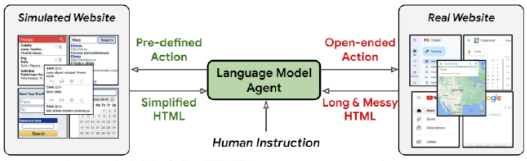
进入强化学习的思想

静态Agent

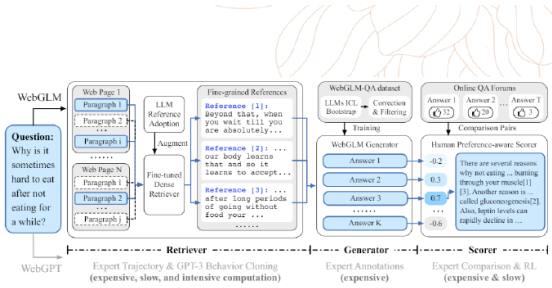
将人类浏览网页的过程拆解为子过程逐模块的使用Agent进行模拟



GopherCite, Teaching language models to support answers with verified quotes, DeepMind, 2022



WebAgent, A Real-World WebAgent with Planning, Long Context Understanding, and Program Synthesis, DeepMind, 2023



WebGLM: Towards An Efficient Web-Enhanced Question Answering System with Human Preference, Liu et al., 2023

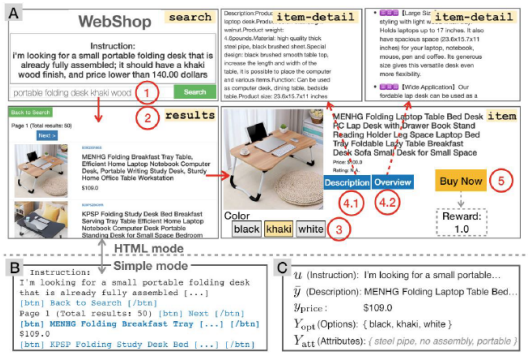
动态Agent

智能体自行决定行为

- Agent自行决定行为

(a) Screenshot from the demonstration interface. (b) Corresponding text given to the model.

WebGPT: Browser-assisted question-answering with human feedback, OpenAI, 2022



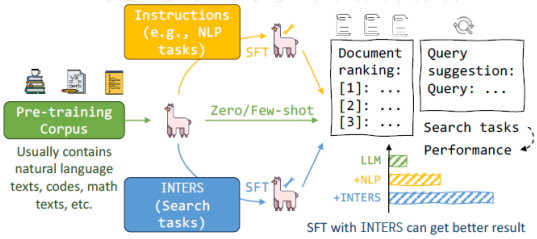
WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents, Yao et al., 2023

6. ACL24 面向信息检索任务的指令微调

INTERS: Unlocking the Power of Large Language Models in Search with Instruction Tuning

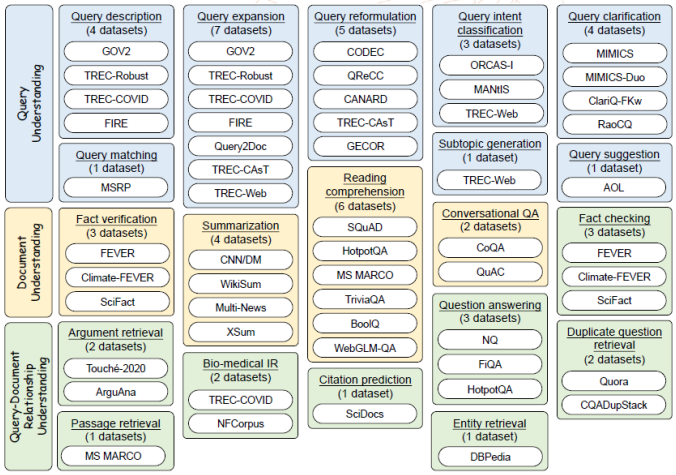
动机

- 大语言模型在NLP任务上表现出了极强的能力
- 指令微调进一步实现了大模型与人类任务的对齐
- 在IR任务上，大模型（基于提示）没有显著优于小模型
- 大模型的预训练中缺乏对IR概念的理解
  - 例如：查询、文档、相关性、用户意图等
- 已有指令微调数据集缺乏IR相关任务
- 解决方法 ⇒ **构建面向IR任务的指令微调数据集**



基本思路

- 分析划分已有IR任务
  - 3个任务类
    - 查询理解
    - 文档理解
    - 查询-文档关系理解
  - 20个任务
  - 43个数据集
- 收集并构建指令微调数据集
- 进行大量实验与分析



Part 2 检索增强的生成大模型

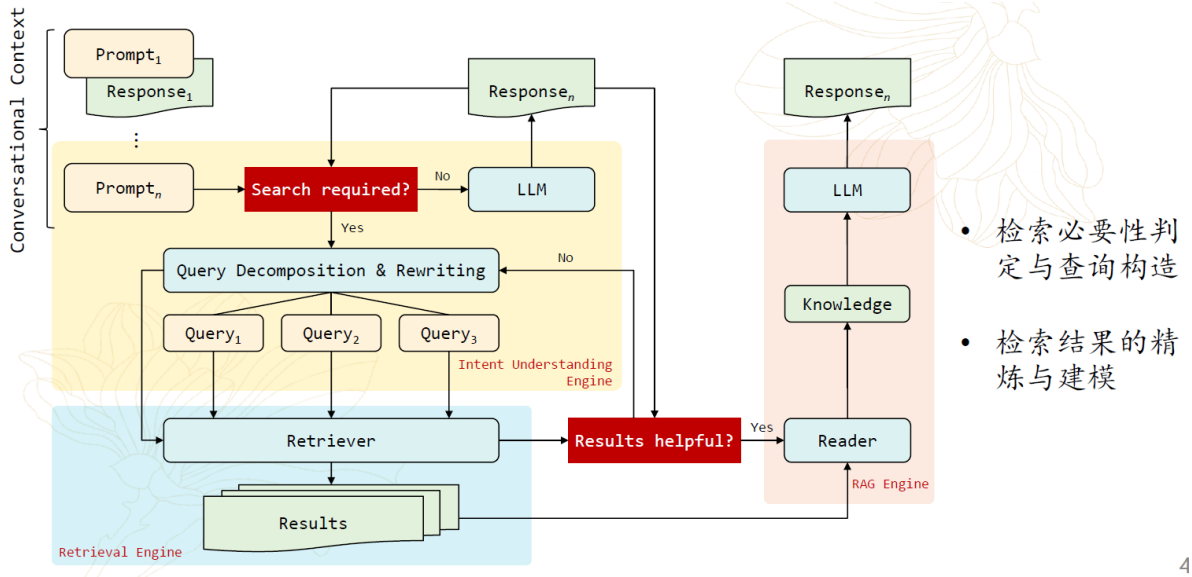
为什么要做检索增强?

- 大模型并不完美 幻觉问题 知识缺陷 时效性问题

未应用检索增强的大模型（左）笼统的套话+乱说，应用检索增强的大模型（右图）能根据查询到的文档来给出问题的答案

2

RAG的基本框架



4

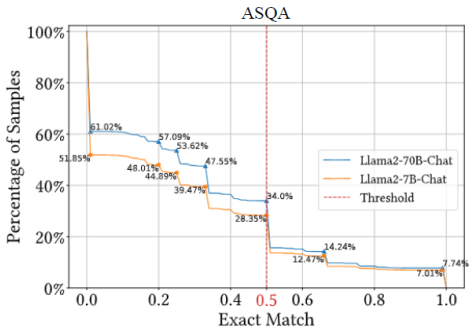
何时需要检索? --检索的必要性判定

ACL24 SlimPLM 代理模型判定检索的必要性

Small Models, Big Insights: Leveraging Slim Proxy Models To Decide When and What to Retrieve for LLMs 2024 ACL

- 检索一定能增强大模型的生成效果么?
  - 无关结果会带来负面的影响
  - 大模型能够掌握的知识不需要检索

- 先前方法：先生成判定结果后在检索生成，这种方式成本高
- 实验发现在模型置信度较高时无论模型规模大小模型预测结果相似
    - 大小模型是否能在知识掌握程度上呈现相似性呢？



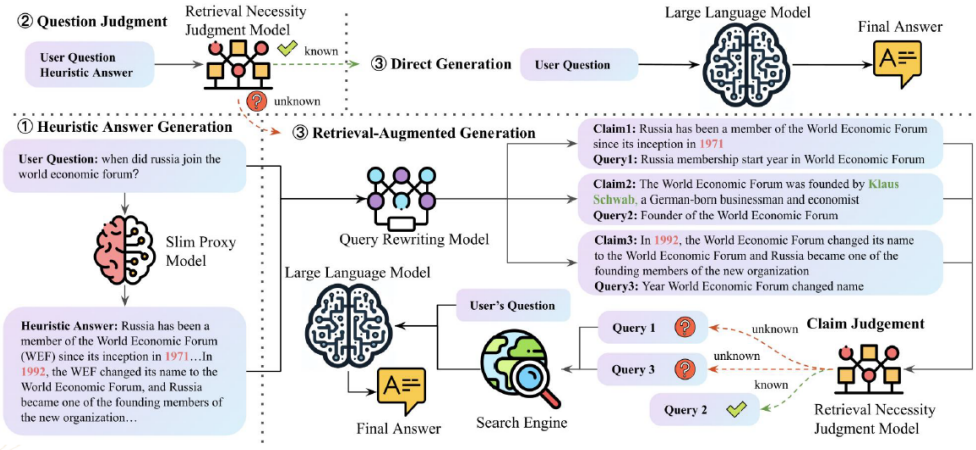
- 回答质量超过当前EM值的样本占总体样本的比例，越高说明当前模型能力越强
- >0.5两条线迅速接近
- 两个模型EM>0.5以上的样本超过80%是一致的

- 目前的LLM用了类似的训练语料
    - 不同LLM在掌握程度高的知识是相当重合的
    - 大小LLM知识能力的差距主要体现在长尾知识上

✓ 用较小参数的语言模型（代理模型）的表现来预测LLM需要做检索的时机！

- 提出使用较小的语言模型作为代理模型，根据代理模型的表现来判定需要做检索的时机

- 代理模型（不做微调）：生成一个回答（Heuristic Answer）
- 检索必要性判断模型（微调后的LLaMA-7b）：决定当前问题是否检索、某个子query是否检索
- 查询重写模型（微调后的LLaMA-7b）：在当前问题需要检索的前提下，生成若干个子query



检索结果如何使用？ --精炼结果的方法

ACL24 BIDER 为大模型精炼有效知识

BIDER: Bridging Knowledge Inconsistency for Efficient Retrieval-Augmented LLMs via Key Supporting Evidence ACL 2024

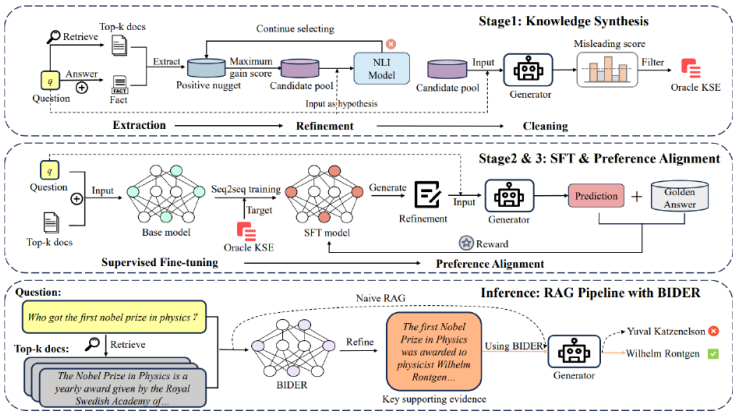
- 造成RAG下降的原因：大语言模型与检索系统之间的知识不一致
    - 检索文档往往冗长且含有噪声
    - LLM无法感知自身的知识边界
- 现有方法过度依赖外部知识（检索）或内部知识（LLM）的一部分而忽略了两者之间的联系
    - 基于困惑度的方法：保留LLM认为有高困惑度的词语
    - 基于模型的方法：保留检索系统判断与正确答案最接近的句子
- 构建模型对检索文档进行精炼，提供LLM最需要的知识

目标：构建模型对检索文档进行精炼，提供LLM在生成时必需的知识

1、构造训练数据
- 根据正确事实的相似度
- 用NLI模型判断对答案的支持程度
- 剔除对LLM有害的知识

2、有监督微调（SFT）
- 学习检索文档到构造的训练数据之间的映射

3、偏好对齐（RL）
- 根据模型偏好进一步调模型
- 考虑LLM的内部知识

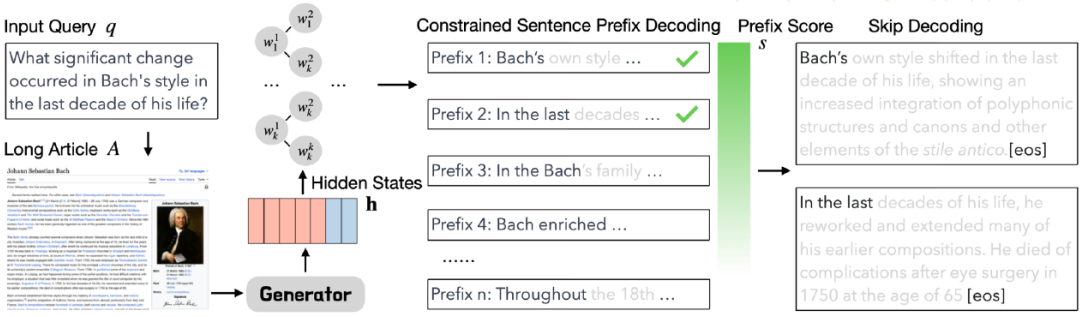


较长的结果如何建模？ --无切块长文本建模方法

ACL24 CFIC 建模更长的检索结果

Grounding Language Model with Chunking-Free In-Context Retrieval ACL 2024

- 检索结果通常比较长，传统RAG处理长上下文时存在局限性
  - LLM无法处理超长文本
  - 长上下文中存在大量的不相关内容
- 已有方法
  - 提高上下文窗口大小直接处理
  - 将长上下文切块和排序，寻找相关内容
- 问题
  - 直接处理长上下文算力要求高，且无法消除长上下文中的噪音
  - 切块破坏了语义的连贯性，导致信息缺失
- 提出一种高效的上下文提炼的方法，不破坏语意连贯性且能高效找到支持回复生成的文本证据



- 不切块，直接基于完整的长文档生成支持回复的文本证据
- 如何保证生成的文本证据是准确的？
  - 通过SFT增强模型处理长上下文的能力，能够更好地适应特定的检索任务
  - 使用受限句子前缀解码，将生成空间限制在句子前缀上，确保生成的文本证据与原始上下文紧密相关
- 如何保证效率？
  - CFIC使用“小而精”的模型处理长上下文，为“大而广”的模型提供准确上下文，生成最终答案
  - CFIC采用跳跃解码策略，一旦确定句子前缀，就跳过中间解码步骤，直接选择最有可能结束的位置，提高解码效率

工具包

arXiv24.5.24 FlashRAG 快速实现RAG方法工具包

FlashRAG: A Modular Toolkit for Efficient Retrieval-Augmented Generation Research

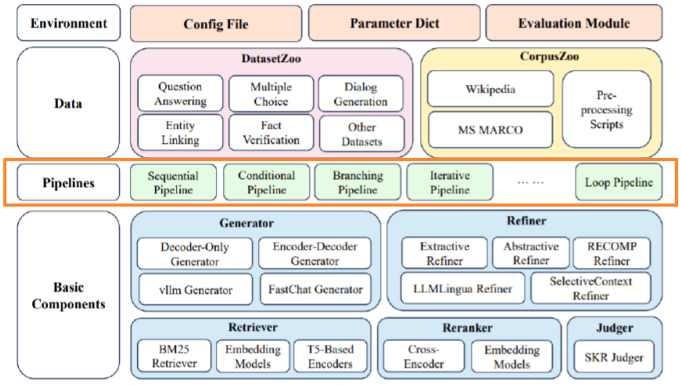
动机：

- RAG系统组件众多，研究人员要花费大量时间在各类工程的实现上
- 现有的RAG工作缺少统一的实现框架，导致复现非常耗时且难以公平比较
- 已有的LangChain LlambIndex工具封装复杂，难以满足定制化研究需求

特点：

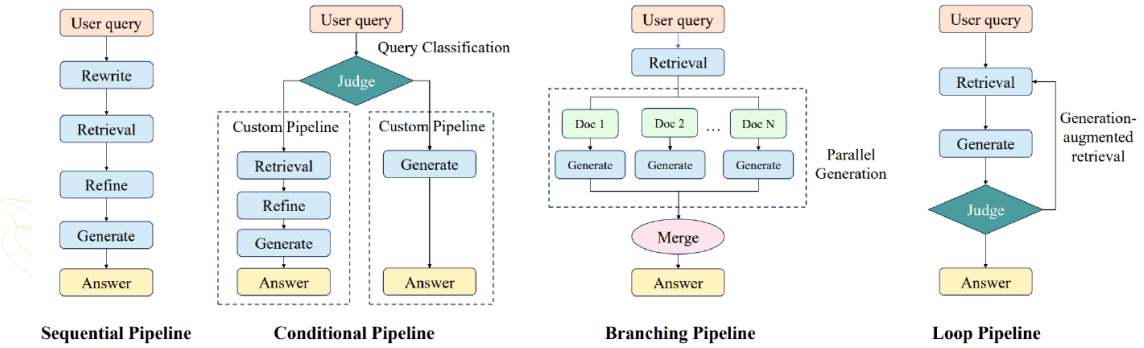
- 模块化RAG框架，包含检索器、生成器、精炼器等多种组件，支持自定义RAG流程
- 目前实现12种RAG工作，轻松在不同设置下评估结果
- 包含32个RAG工作中常用数据集，并预处理为统一的格式
- 包含多种辅助脚本，包含Wikipedia预处理分块、索引构建、检索结果预准备等

  - 基于基础组件构建Pipeline实现常用RAG流程
  - 涵盖32种常用数据集
  - 大多数基于维基百科作为知识源



Task	Dataset Name	Knowledge Source	# Train	# Dev	# Test
QA	NQ [38]	Wiki	79,168	8,757	3,610
	TriviaQA [39]	Wiki & Web	78,785	8,837	11,313
	PopQA [40]	Wiki	/	/	14,267
	SQuAD [41]	Wiki	87,599	10,570	/
	MSMARCO-QA [42]	Web	808,731	101,093	/
	NarrativeQA [43]	Books, movie scripts	32,747	3,461	10,557
	WikiQA [44]	Wiki	20,360	2,733	6,165
	WebQuestions [45]	Google Freebase	3,778	/	2,032
	AmbigQA [46, 38]	Wiki	10,036	2,002	/
	SIQA [47]	-	33,410	1,954	/
	CommenseQA [48]	-	9,741	1,221	/
	BoolQ [49]	Wiki	9,427	3,270	/
	PIQA [50]	-	16,113	1,838	/
	Fermi [51]	Wiki	8,000	1,000	1,000
Multi-Hop QA	HotpotQA [52]	Wiki	90,447	7,405	/
	2WikiMultiHopQA [53]	Wiki	15,000	12,576	/
	Musique [54]	Wiki	19,938	2,417	/
	Bamboogle [32]	Wiki	/	/	125
Long-Form QA	ASQA [55]	Wiki	4,353	948	/
	ELI5 [56]	Reddit	272,634	1,507	/
Multiple-Choice	MMLU [35, 36]	-	99,842	1,531	14,042
	TruthfulQA [57]	Wiki	/	817	/
	HellaSwag [58]	ActivityNet	39,905	10,042	/
	ARC [59]	-	3,370	869	3,548
	OpenBookQA [37]	-	4,957	500	500
Entity-linking	AIDA CoNLL-YAGO [60, 61]	Wiki & Freebase	18,395	4,784	/
	WNED [62, 61]	Wiki	/	8,995	/
Slot filling	T-REx [63, 61]	DBPedia	2,284,168	5,000	/
	Zero-shot RE [64, 61]	Wiki	147,909	3,724	/
Fact Verification	FEVER [65, 61]	Wiki	104,966	10,444	/
Dialog Generation	WOW [66, 61]	Wiki	63,734	3,054	/
Open-domain Summarization*	WikiAsp [67]	Wiki	300,636	37,046	37,368

- 目前已支持常见的四种不同流水线的RAG工作



数据集

WebBrain 面向RAG的通用数据集

WebBrain: Learning to Generate Factually Correct Articles for Queries by Grounding on Large Web Corpus arXiv2304

- 现有的RAG数据集，尤其是训练集不足
  - 已有工作多采用open-domain QA 作为训练和测试集
  - 人为使用检索器构建训练集合，检索和生成文本的关联性缺乏保障
  - 难以判断是否参考了检索结果
- 提出基于维基百科的文本及其引用构建大规模数据集
  - 包括对维基百科引用链接进行标注

- 超大规模原始数据集（2.8TB）
  - 支撑预训练、微调等多种研究
- 高质量数据源（Wikipedia）
- 包含引用标号，可解释性良好
- 已清洗、抽取并划分检索与生成集合，支持多种研究

	WebBrain-R	WebBrain-G
# Queries	2.74M	12.32M
# Ref. passages	3.20M	12.61M
# Tokens / Query	3.2	2.9
# Tokens / Passage	237.5	250.0
# Tokens / Target	-	108.6
# Training	4.46M	12.30M
# Validation	0.2M	0.5M
# Test	88,935	24,546

Dataset	# Wiki Pages	# Refs	Status	Storage Size
WikiSum (Liu et al., 2018)	2.3M	87M	Need crawling	300GB
WikiCatSum (Perez-Beltrachini et al., 2019)	0.17M	23.5M	Ready	4.8GB
Hiersumm (Liu & Lapata, 2019)	1.66M	-	Ready	6.9GB
WebBrain-Raw	14.86M	259.5M	Ready	2.8TB

DomainRAG 特定领域RAG评测

DomainRAG: A Chinese Benchmark for Evaluating Domain-specific Retrieval-Augmented Generation ACL 2024、

动机：

- RAG能够有效解决LLM的各种限制，例如幻觉和知识实时更新的困难
- 目前的研究往往依赖于维基百科等一般知识源来评估模型解决常识性问题的能力，然而RAG在LLM难以涵盖专业知识的场景和特定领域中的应用也很重要

方法：

- 使用特定领域的语料库和问题对于评估LLM有效利用来自这些特定领域的外部知识来解决专家问题的能力至关重要
- 总结综合评价RAG模型的六个重要能力，并以人大招生为应用场景构建了评估这些能力的数据集

Future Work

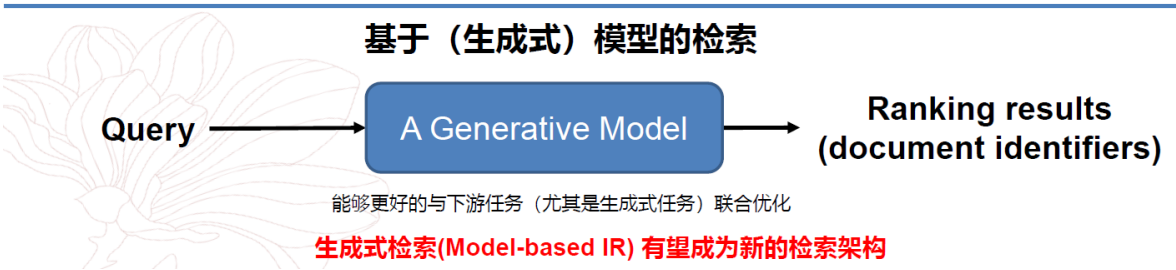
- 更加精准的查询分解与改写
- 对话式RAG的进一步探索
- 面向RAG的训练 （预训练、指令微调）
- 长窗口与RAG之间的关联
- RAG系统的评估fang'fa

Part 3 生成式文档检索

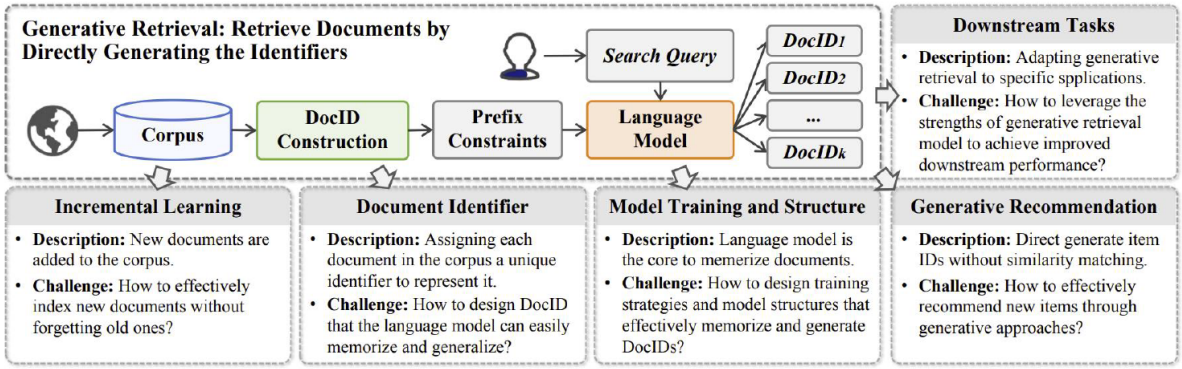
From Matching to Generation: A Survey on Generative Information Retrieval arXiv2404

一个核心的问题：能否直接通过大模型完成文档的检索/召回?

- 大模型并没有检索能力
- 大模型瞎编烂造的能力在检索相关问题上体现的淋漓尽致
- 大模型需要定向微调才能实现检索能力



生成式检索模型目前面临的问题



增量学习问题

- 文档的动态更新，大模型怎么去适配?
- 如何处理海量的文档?
- 如何将文档嵌入到模型中?

文档标识定义

- 如何定义DockID能够让模型更轻松的记忆和泛化

训练策略和模型架构

- 如何设计架构和策略来让模型更高效的记忆和泛化海量文档

生成答案

- 如何通过查询到的文档高效的生成答案

经典工作

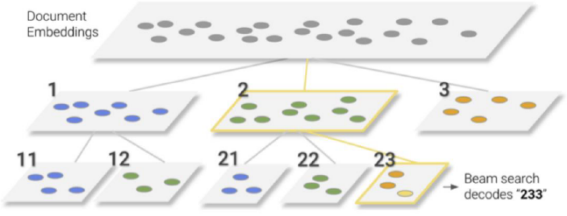
DSI (Google)

提出一种分层的文档编码方案

Transformer Memory as a Differentiable Search Index NeruIPS 2022

- Indexing Method

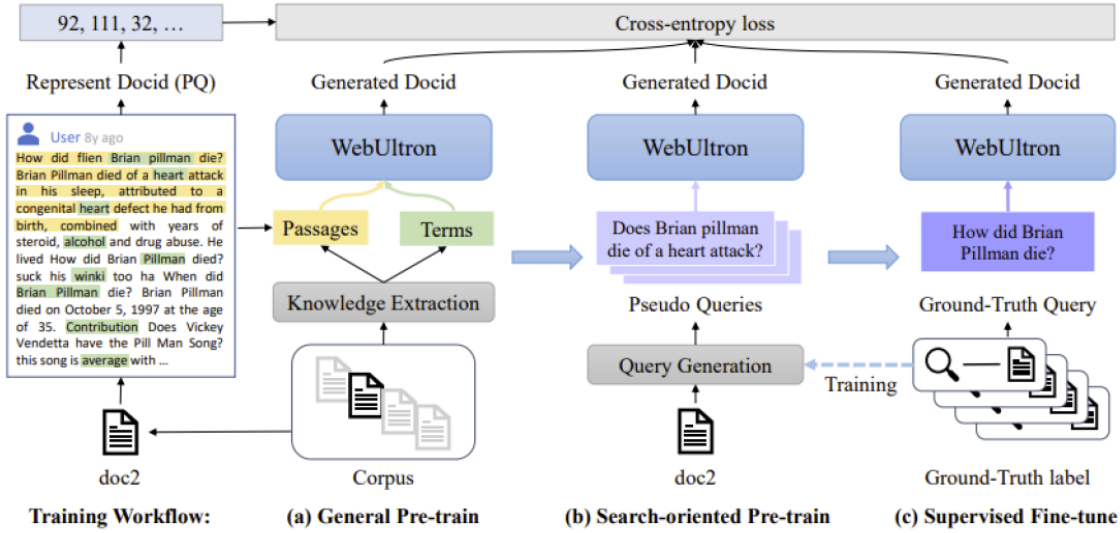
  `doc_tokens` → `docid` `docid` → `doc_tokens`
- What to index:
  - First L tokens; set of terms; a single contiguous chunk of k tokens
- DocId: Atomic Identifiers; naively structured identifiers; Semantically Structured Identifiers



WebUltron (renda)

给出一种三阶段训练框架

WebUltron: An Ultimate Retriever on Webpages Under the Model-Centric Paradigm 2023 IEEE Transactions on Knowledge and Data Engineering



NCI (MicroSoft)

A Neural Corpus Indexer for Document Retrieval NeruIPS 2022

提出一种神经语料库检索器，序列到序列的网络

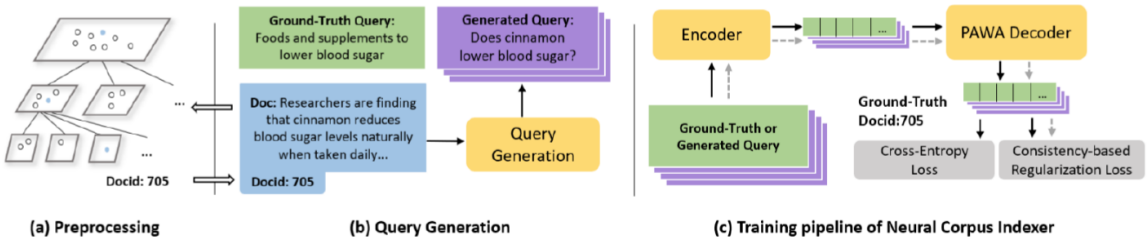


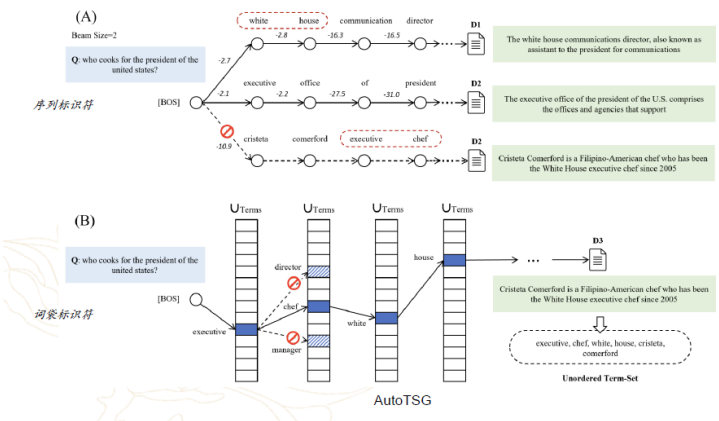
Figure 1: Overview of Neural Corpus Indexer (NCI). (a) Preprocessing. Each document is represented by a semantic identifier via hierarchical k-means. (b) Query Generation. Queries are generated for each document based on the content. (c) The training pipeline of NCI. The model is trained over augmented <*query*, *docid*> pairs through a standard transformer encoder and the proposed Prefix-Aware Weight-Adaptive (PAWA) Decoder.

跳出Sequence范式，词袋模型 (renda)

Generative Retrieval via Term Set Generation SIGIR 2024

文档ID是序列化的形式，解码错一步则全错

提出基于词袋的方案，生成词袋的顺序构成了文档标识符



词袋选取

- 通过深度词权重模型，抽取最重要的N个词

受限制解码

- 为每个文档存储N!个序列是不可行的。
- 设计了基于倒排索引的受限制解码方式。

模型优化

- 无法提前从N!个序列中选取最优的序列作为学习目标
- 迭代式优化：从一个初始序开始，优化模型参数，收敛后令模型从每个文档的N!个序列中采样最优的作为下一轮学习目标

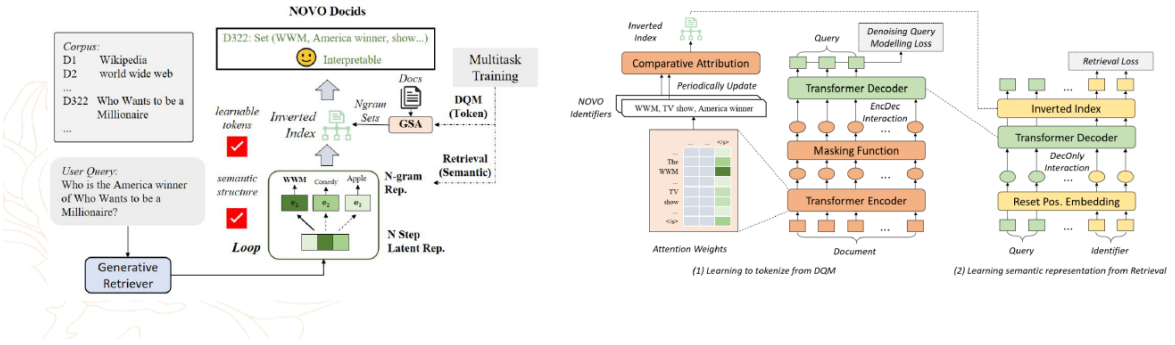
可学习的文档标识符 (renda)

NOVO: Learnable and Interpretable Document Identifiers for Model-Based IR CIKM 2023

现有的文档ID基于Encoder独立完成，与Decoder无关，存在Gap，提出了一种可学习的文档标识符方案

可学习的文档标识符NOVO

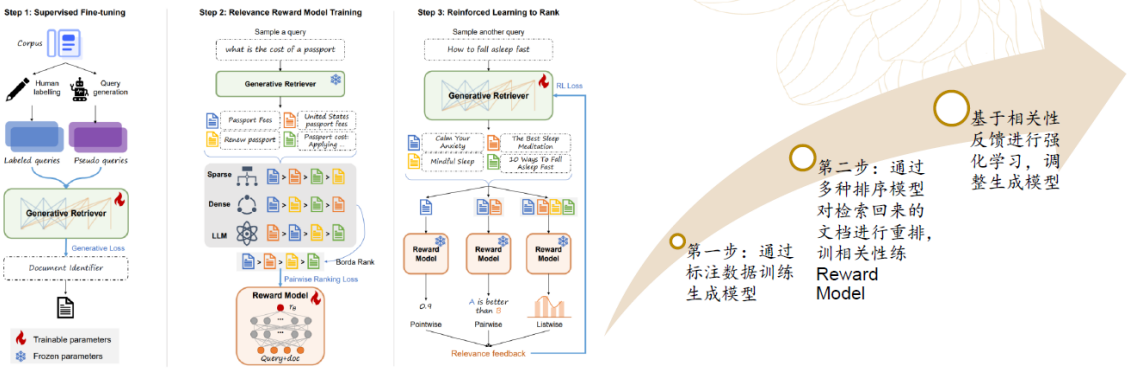
- 由N元组集合组成，随模型生成过程，通过倒排索引召回相应文档
- N元组通过全局自注意力(GSA)信息从文档中获取，并通过两种任务(DQM, Retrieval)优化标识符令牌及语义，并与检索任务对偶优化



相关性强化的生成式检索模型 （renda）

Enhancing Generative Retrieval with Reinforcement Learning from Relevance Feedback EMNLP 2023

引入基于相关性反馈的强化学习来让模型理解相关性



生成式检索与其他生成任务的融合 （renda）

UniGen: A Unified Generative Framework for Retrieval and Question Answering with Large Language Models AAAI 2024

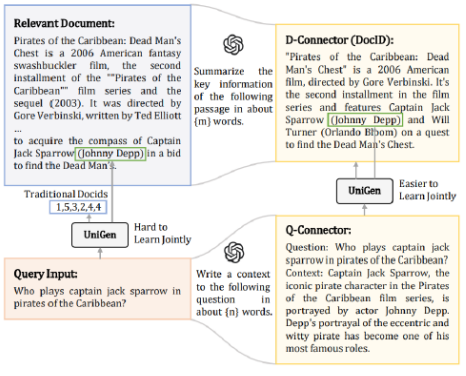


Figure 3: An example of generating LLM-based connectors from the query side and document side, with the labeled answer highlighted in the green box.

$$\mathcal{L}_{\text{retr}} = -\sum_{i=1}^{t} \log f_{\text{retr}}(d_i | d'_{<i}, q; \theta, \phi). \quad (2)$$

$$\mathcal{L}_{\text{qa}} = -\sum_{i=1}^{T'} \log f_{\text{qa}}(a_i | a_{<i}, q; \theta, \mu). \quad (4)$$

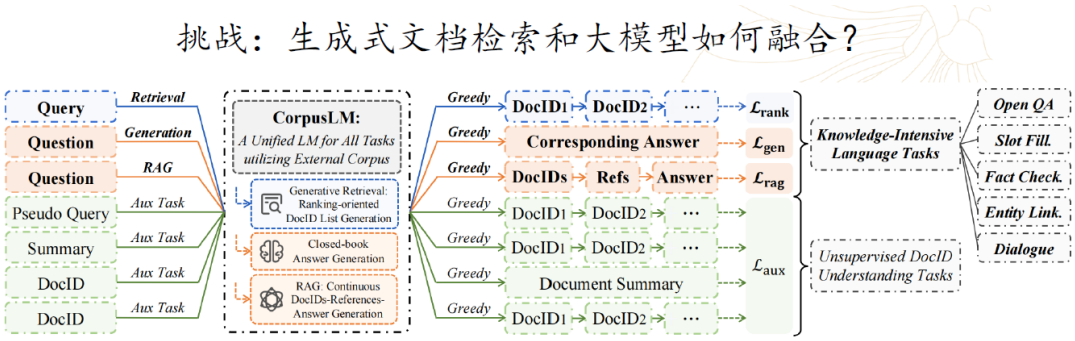
$$\mathcal{L}'_{\text{retr}} = -\sum_{i} \log f_{\text{retr}}(d_{c_i} | d_{c_{<i}}, q_c; \theta, \phi), \quad (5)$$

$$\mathcal{L}'_{\text{qa}} = -\sum_{i} \log f_{\text{qa}}(a_i | a_{<i}, q_c; \theta, \mu), \quad (6)$$

$$\mathcal{L} = \lambda \mathcal{L}'_{\text{retr}} + (1-\lambda)\mathcal{L}'_{\text{qa}}, \quad (7)$$

CorpusLM (renda)

CorpusLM: Towards a Unified Language Model on Corpus for Knowledge-Intensive Tasks SIGIR 2024



- **统一的语言模型：**集成生成式检索、闭卷生成和检索增强生成（RAG），辅助知识密集型任务。
- **面向排名的DocID列表生成策略：**贪婪解码生成DocID排序。
- **RAG导向的连续生成策略：**连续解码方法。
- **无监督DocID理解任务：**无监督的DocID理解任务，加深模型对DocIDs背后含义的理解，以进一步提高了CorpusLM的检索和生成性能。

Report 3 大模型时代的通用向量检索

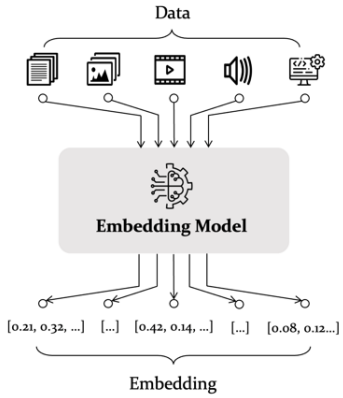
北京智源研究院 刘政

01 什么是语义向量模型

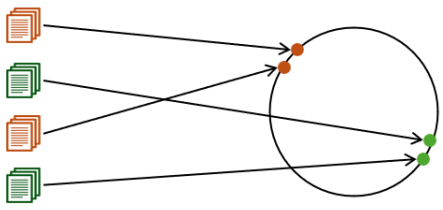
向量模型：将任意数据转化为高维空间中稠密向量的计算机模型

重要属性：**向量相似性**要与**数据相似性**保持一致

这里的相似性计算并不严格，不受三角不等式约束



Data2Vec

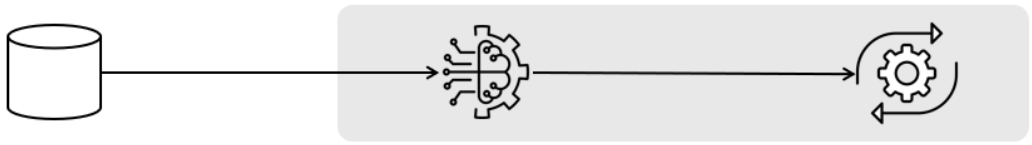


relevant data -> similar embeddings

向量模型应用：

- 信息检索系统
- 比较数据的语义关联，数据聚类去重等
- 向量数据库：不再追求相似性最高的目标向量，近似最近邻计算节约开销

02 向量学习的基本模式



Training Data

massive, diverse, and high-quality datasets

Embedding Model

expressive, well pre-trained foundation models

Learning Algorithm

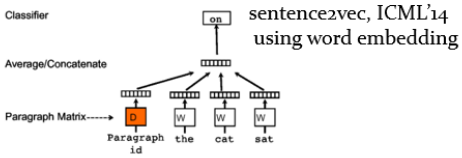
task-oriented, data-efficient learning algorithms

1. 训练数据

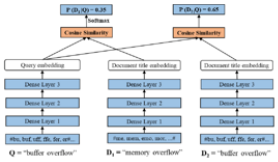
2. 模型

Foundation model for embedding

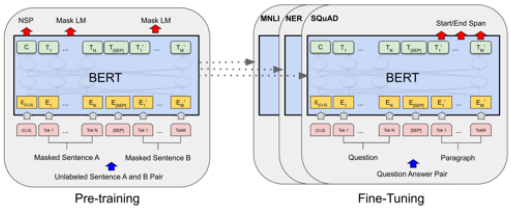
- Almost any DNNs can be naturally applied as the embedding model (Word2Vec, LSTM, etc.)
- Not until the arrival of pre-trained LMs (e.g., BERT) that embedding model really works
- Advantage of PLMs: 1) transformer architecture, 2) pre-training



sentence2vec, ICML'14
using word embedding



dssm, cikm'13
c-dssm, www'14
using conv and mlp



Transformer: strong context-awareness, free of inductive bias
Pre-trained LM: deep understanding of language, and hence common-sense semantic meaning of data

- 几乎所有的DNN模型都可以作为向量模型
- 但真正让向量模型变得可用的是，以Transformer为基础的预训练模型，其本质是“大”
- Transformer的网络架构可以做的足够大，让模型由足够的容量去建立强大的表达能力
- 预训练的规模可以足够大，让模型充分学习海量数据中的知识

主流的预训练算法对向量检索而言是不是最优的?

- 修改模型训练目标：让embedding直接体现在训练的优化之中
- 优化embedding对输入文本语义的表达能力：先前的单个词的预测不合适，简单的转为预测目标文本的全部词汇

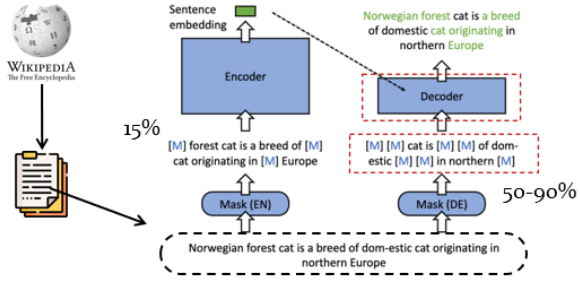
基于此：提出RetroMAE

RetroMAE: Pre-Training Retrieval-oriented Language Models Via Masked Auto-Encoder EMNLP 2022

- 从任意的无标签语料，比如Wikipedia中采样一段文本，将其编码成embedding之后再将其解码出原始的文本
- 非对称的编码器解码器结构，非对称的掩码比率，使得模型输出的embedding可以更加充分的从训练数据中得到学习优化

The design of RetroMAE (Retrieval-Oriented Pre-training Based Mask Auto-Encoder)

- **High-level framework**: 1) sample a sentence, 2) mask and encode, 3) decode the clean input
- **Concrete process**: 1) asymmetric structure, full-scale encoder (12L), small-scale decoder (1L), 2) asymmetric masks, encoder: 15%, decoder: 50-90%
- **Data**: purely based on unlabeled data (Wikipedia corpus), same as general pre-training



Why asymmetric?

- Small decoder is time efficient
- Small decoder increases the hardness of learning
- Aggressive decoding mask enables high sample efficiency (training loss generated by masked tokens)

3. 训练算法

03 BGE模型的开发与实践

04 大语言模型与信息检索

Report 4 检索增强大模型技术探索与思考

中国科学院计算技术研究所 庞亮

01 检索视角下的检索增强

02 大模型视角下的检索增强

03 交互视角下的检索增强

04 信息回路视角下的检索增强

Report 5 When Search Engine Meets LLMs: Opportunities and Challenges

微软亚洲研究院 王亮

LLMs如何帮助现有的搜索技术栈

LLMs是强大的数据生成器

搜索引擎如何增强LLMs

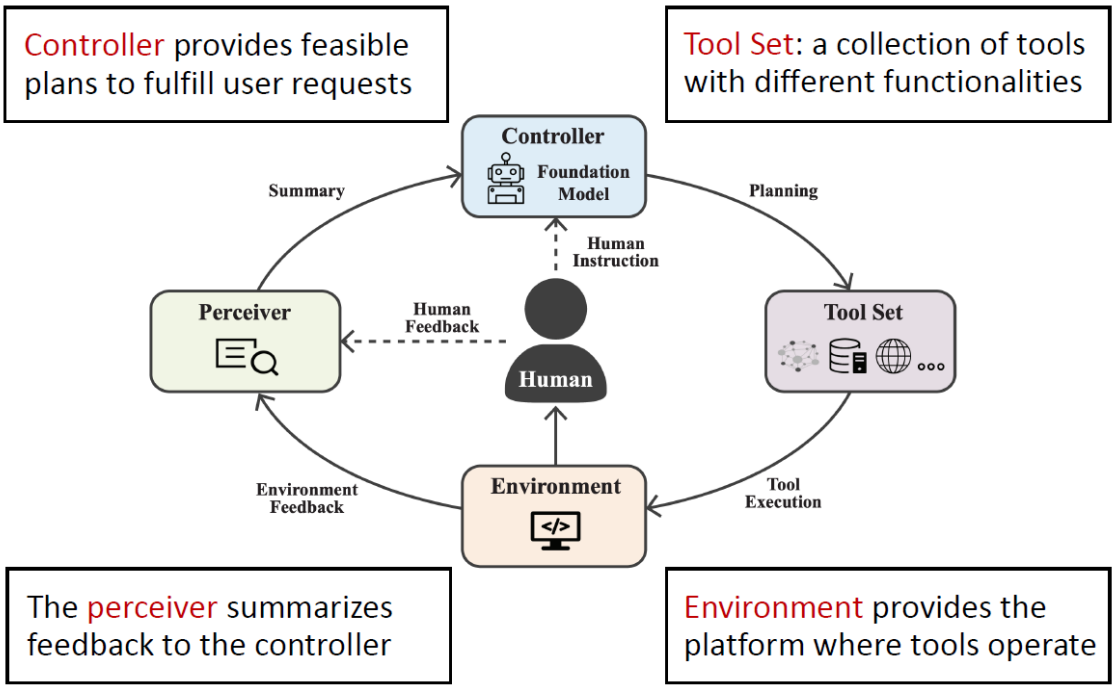
LLMs会取代搜索引擎么

未来展望

Report 6. LLM-Based Tool Learning and Autonomous Agents

中国人民大学 高瓴人工智能学院 林衍凯

Survey: Tool Learning with Foundation Models arXiv2304



- 控制器提供可行的计划来满足用户的请求
- 工具集合为一系列拥有不同功能的集合

- 环境提供工具操作的平台
- 感知者向控制器总结反馈信息

- Controller $\mathcal{C}$ generates a plan a_t

Feedback History Instruction

$$p_C(a_t) = p_{\theta_C}(a_t \mid x_t, \mathcal{H}_t, q)$$

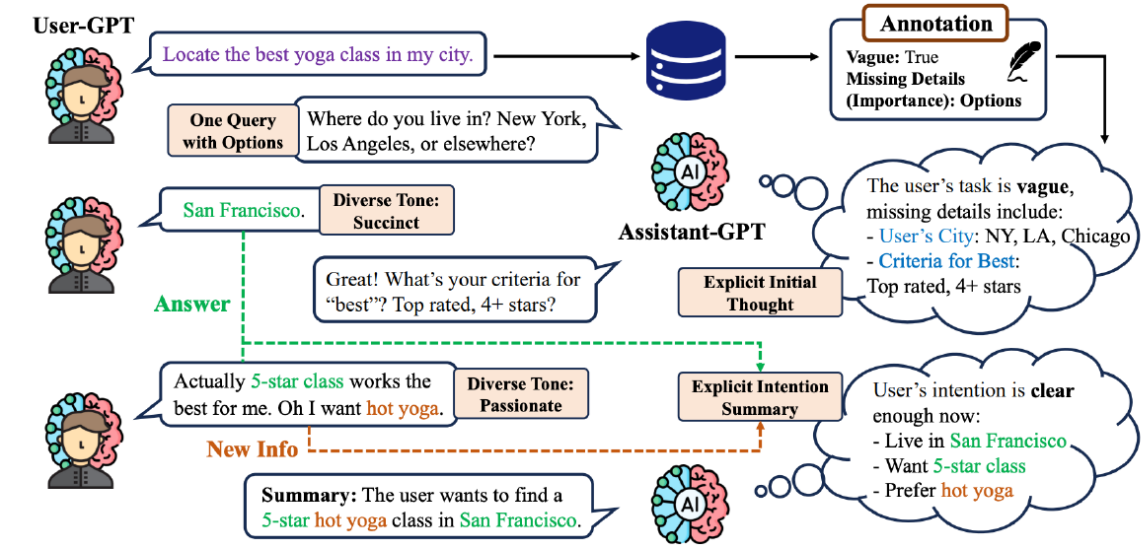
- Problem
  - Intent Understanding: understand the user task intent
  - Planning: divide the user query into sub-tasks
  - Tool Use: use the appropriate tool to solve sub-task
  - Memory: manage the working history

Intent Understanding: understand the user task intent

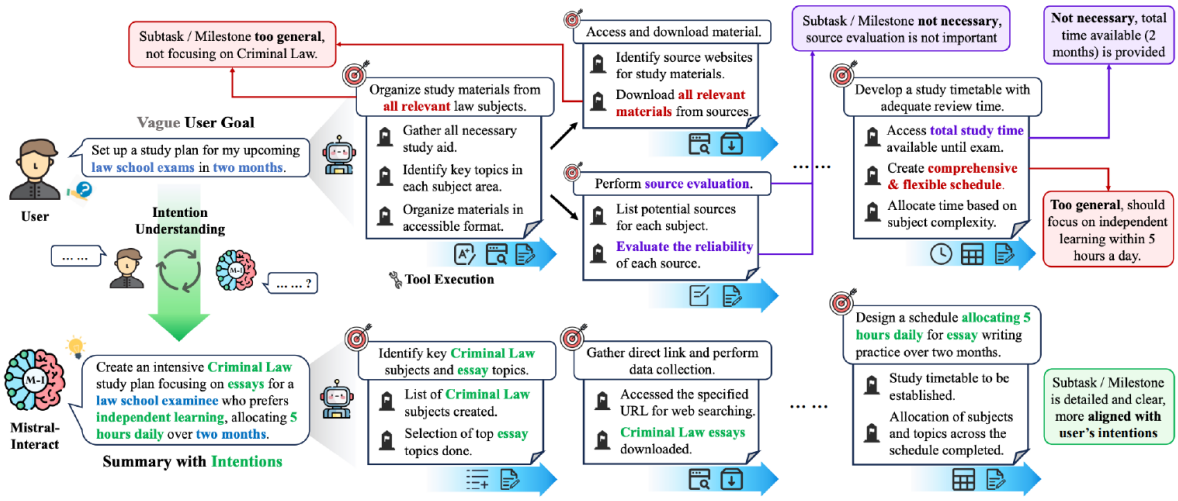
意图理解，理解用户的任务意图

难点：理解用户的模糊指令

将任务传递给下级执行之前，agent应当主动且明确的向用户询问缺失的细节



一个例子：

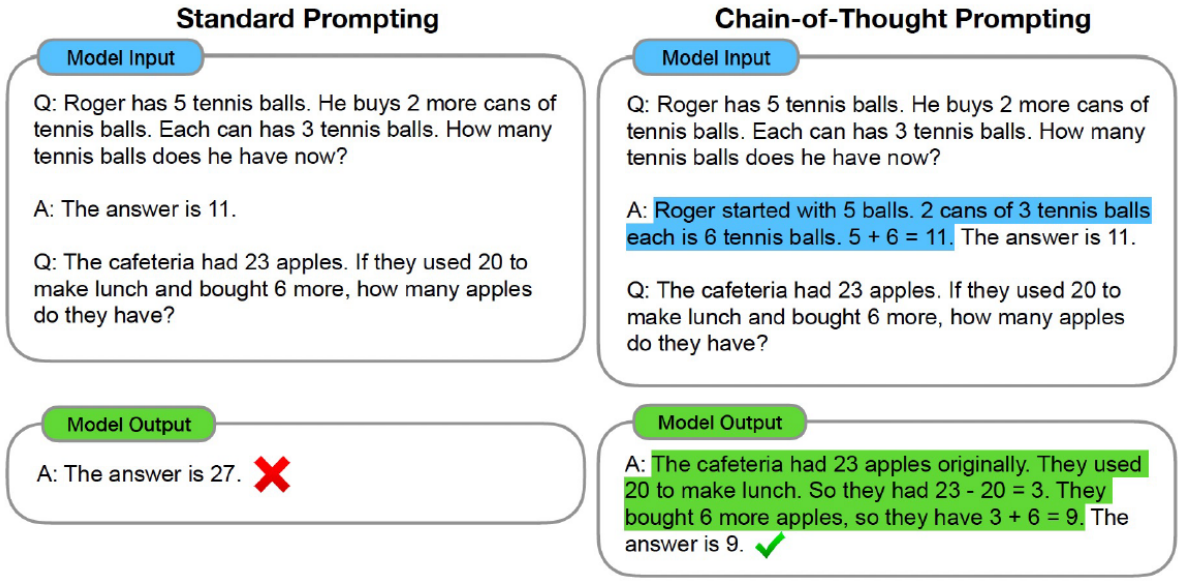


Planning: divide the user query into sub-tasks

将用户的查询分解为多个子问题来分别处理

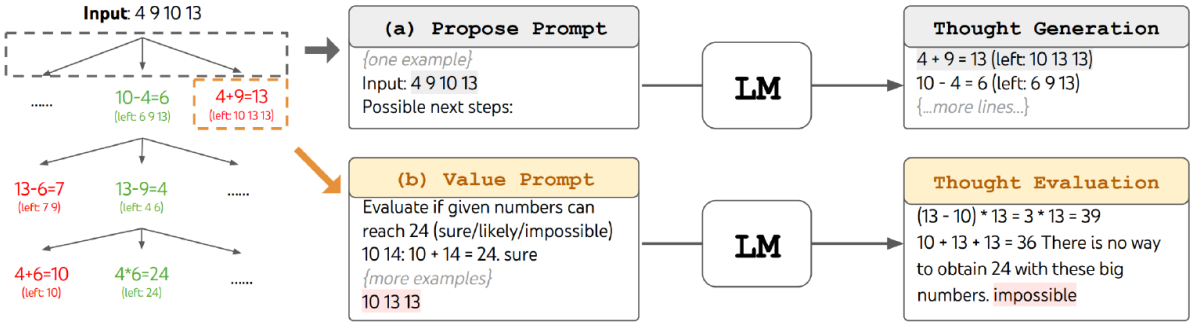
Chain of Thought (CoT)

思维链方法：给出推导过程的思维过程



Tree of Thought (ToT)

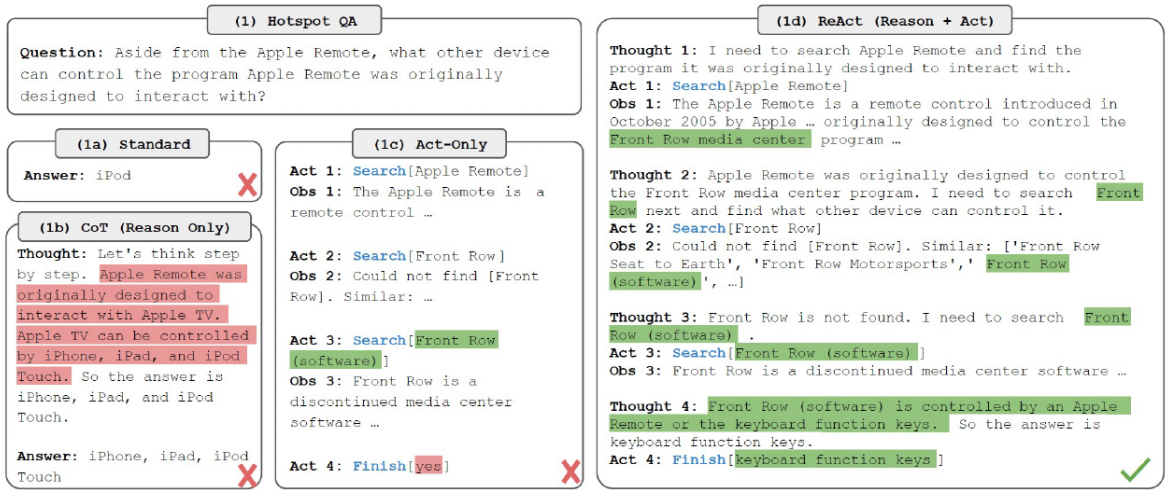
思维树方法：给出推导过程的搜索树



ReAct 2023 ICLR

ReAct: Synergizing Reasoning and Acting in Language Models

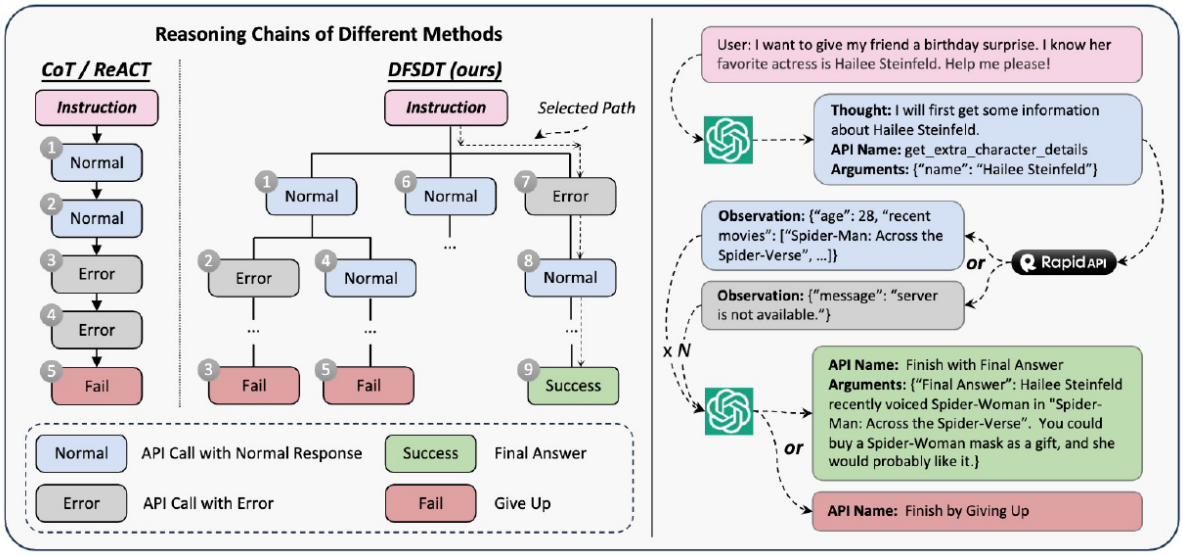
CoT推理帮助模型更新行动计划以及处理异常，而行动允许它与外部源（例如知识库或环境）进行交互，以收集更多信息



DFSDT 深度优先搜索的决策树

ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs arXiv2307

训练了 API 检索器来为每条指令推荐适当的 API



Human-Agent Collaboration 人类-智能体协作

有些问题全部由智能体来代理性能并不佳，但是将其中很小一部分交给人类完成性能能够得到大幅提升，引入人机协作问题

- Formulate as an RL problem:

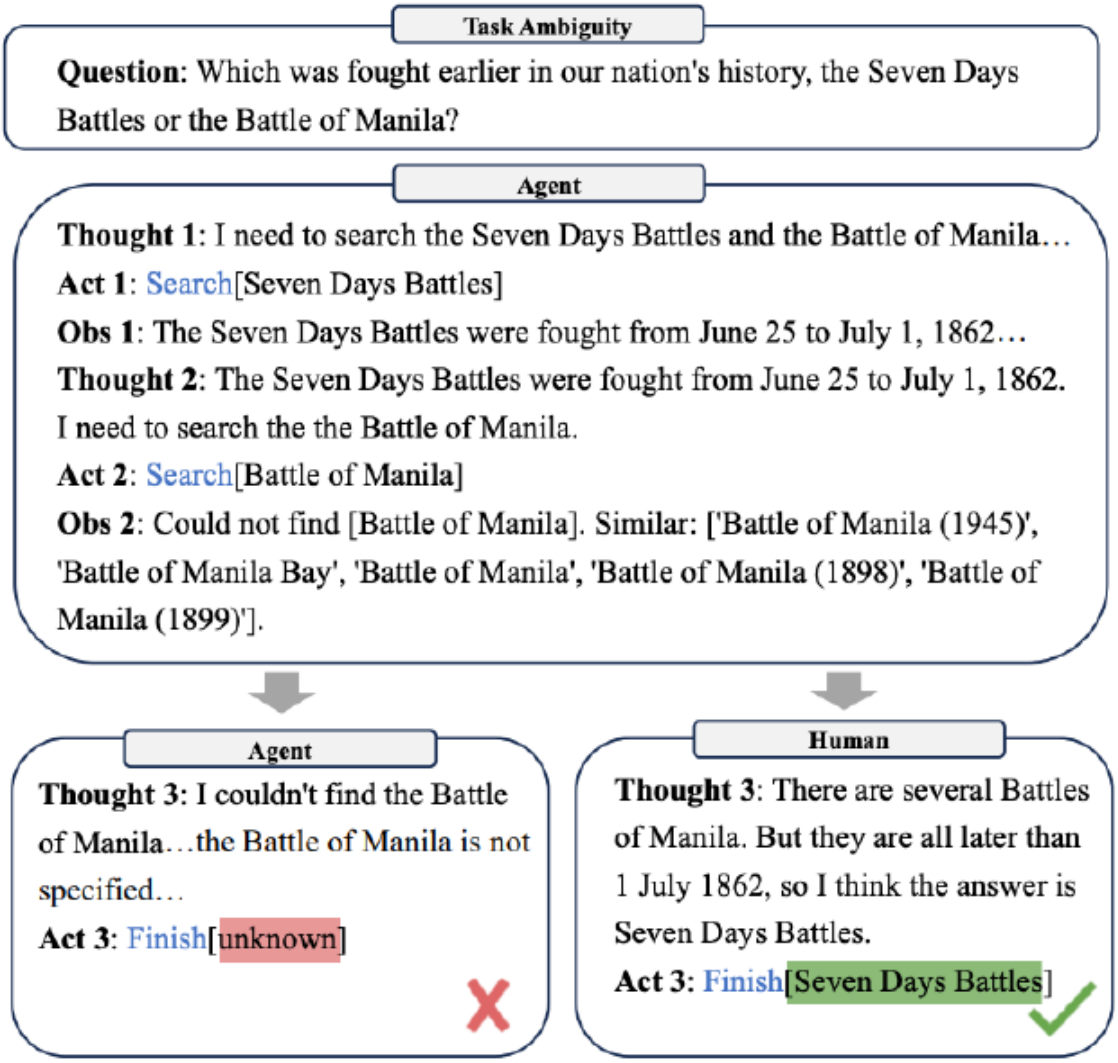
$$\mathcal{J}(\pi_\theta) = \mathbb{E}[\frac{\pi_\theta(a|s)}{\pi_{\text{beh}}(a|s)} A(s,a)],$$

$$A(s,a) = R(s,a) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} R(s,a')$$

- Balance the maximization of task performance and the cost of human intervention

$$R(s,a) = T(s,a) - \lambda C(s,a)$$

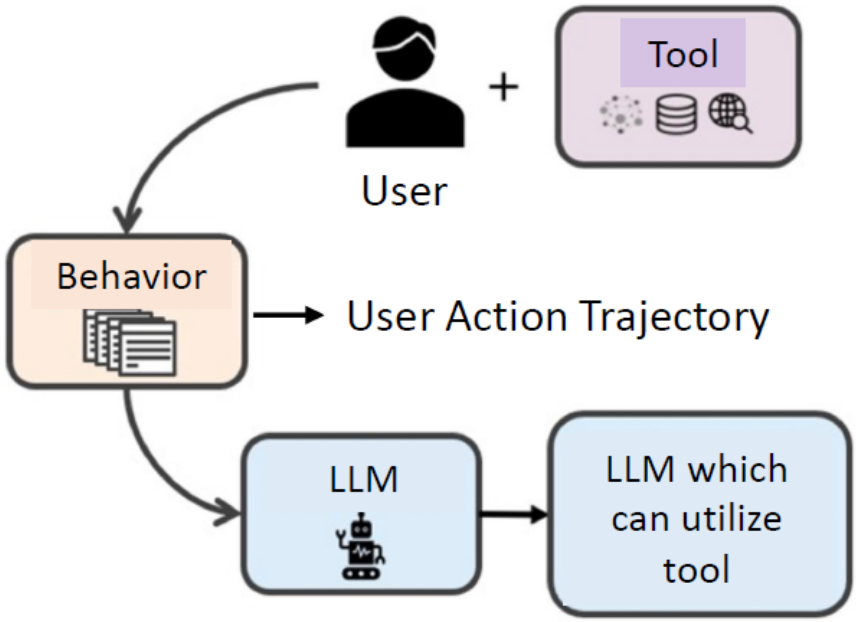
λ来控制人类的参与程度



Tool Use: use the appropriate tool to solve sub-task

模仿学习：一个最简单的学习范式。

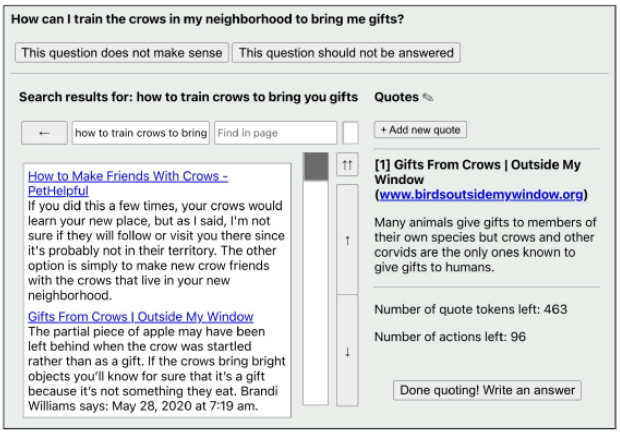
通过记录人类使用工具的行为数据，让大模型来模拟人类的行为来了解工具



WebGPT

WebGPT: Browser-assisted question-answering with human feedback arXiv2112

- 模仿类使用搜索引擎的行为
- 监督微调 + 强化学习
- 只需 6,000 条标注数据




```
♦Question
How can I train the crows in my neighborhood to bring me gifts?

♦Quotes
From Gifts From Crows | Outside My Window (www.birdsoutsidemywindow.org)
> Many animals give gifts to members of their own species but crows and
other corvids are the only ones known to give gifts to humans.

♦Past actions
Search how to train crows to bring you gifts
Click Gifts From Crows | Outside My Window www.birdsoutsidemywindow.org
Quote
Back

♦Title
Search results for: how to train crows to bring you gifts

♦Scrollbar: 0 - 11
♦Text
【0†How to Make Friends With Crows - PetHelpful†pethelpful.com】
If you did this a few times, your crows would learn your new place, but
as I said, I'm not sure if they will follow or visit you there since it's
probably not in their territory. The other option is simply to make new
crow friends with the crows that live in your new neighborhood.

【1†Gifts From Crows | Outside My Window†www.birdsoutsidemywindow.org】
The partial piece of apple may have been left behind when the crow was
startled rather than as a gift. If the crows bring bright objects you'll
know for sure that it's a gift because it's not something they eat.
Brandi Williams says: May 28, 2020 at 7:19 am.

♦Actions left: 96
♦Next action
```

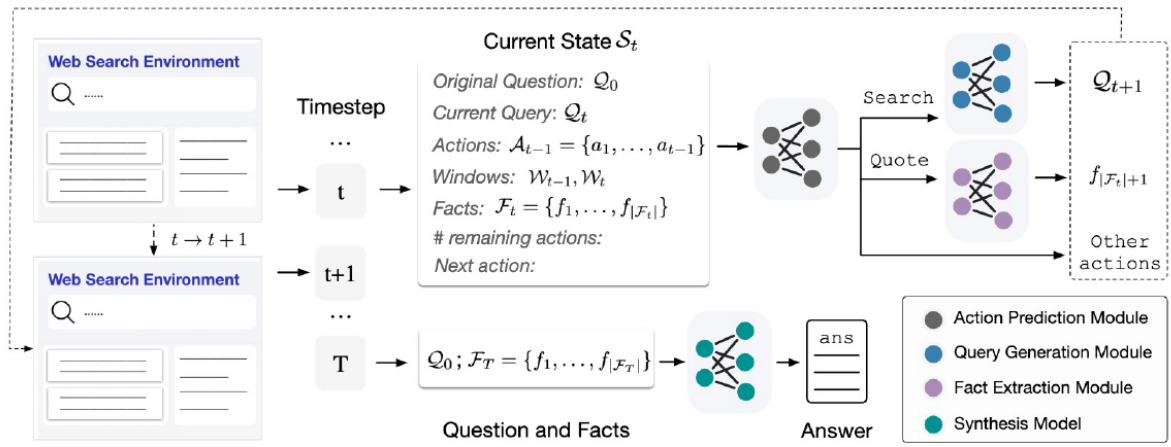
WebCPM: Chinese WebGPT

Interactive web search for Chinese long-form question answering. ACL 2023

中文版的WebGPT

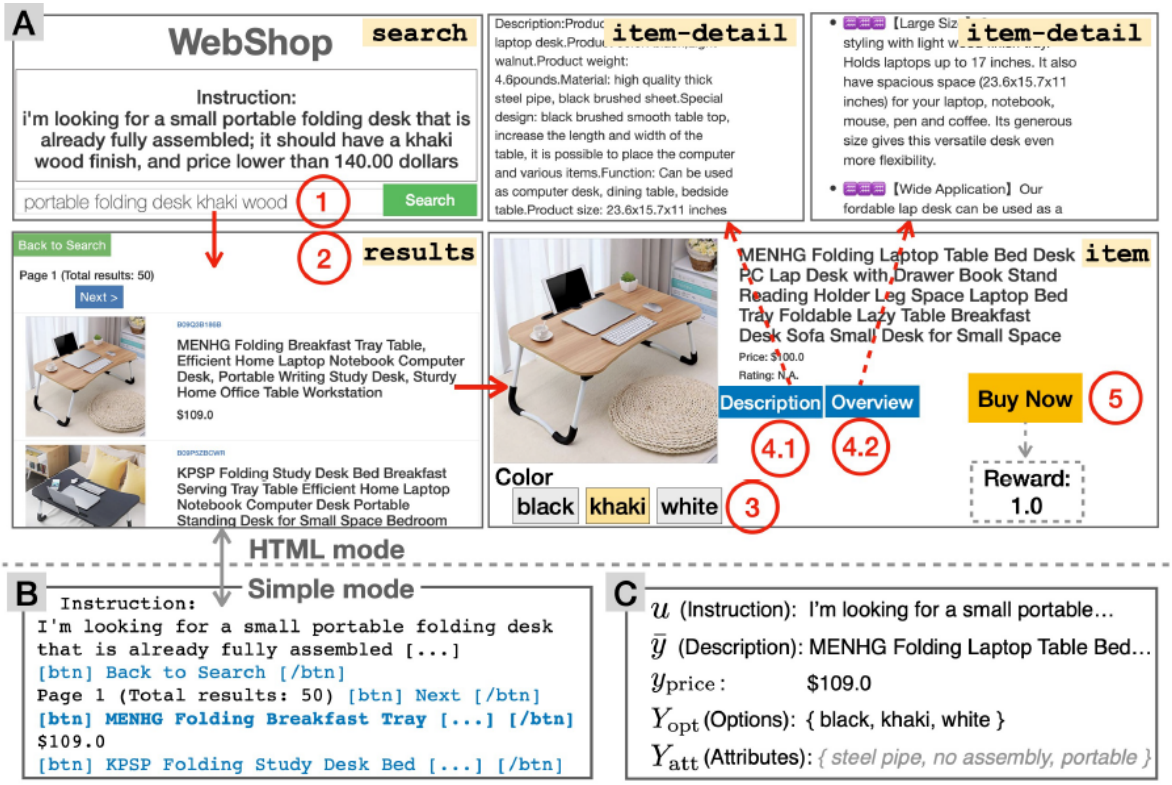
Action Name	Functionality
Search <query>	Call Bing search with <query>
Go Back	Return to the previous window
Load Page <1>	Load the details of page <1>
Load Page <2>	Load the details of page <2>
Load Page <3>	Load the details of page <3>
Scroll Up	Scroll up for a pre-set stride
Scroll Down	Scroll down for a pre-set stride
Quote <content>	Extract <content> from the current page as a supporting fact
Merge	Merge two facts into a single fact
Finish	End the search process

在每个步骤中，搜索模型都会执行操作以收集证据，并将其发送给大模型以生成答案



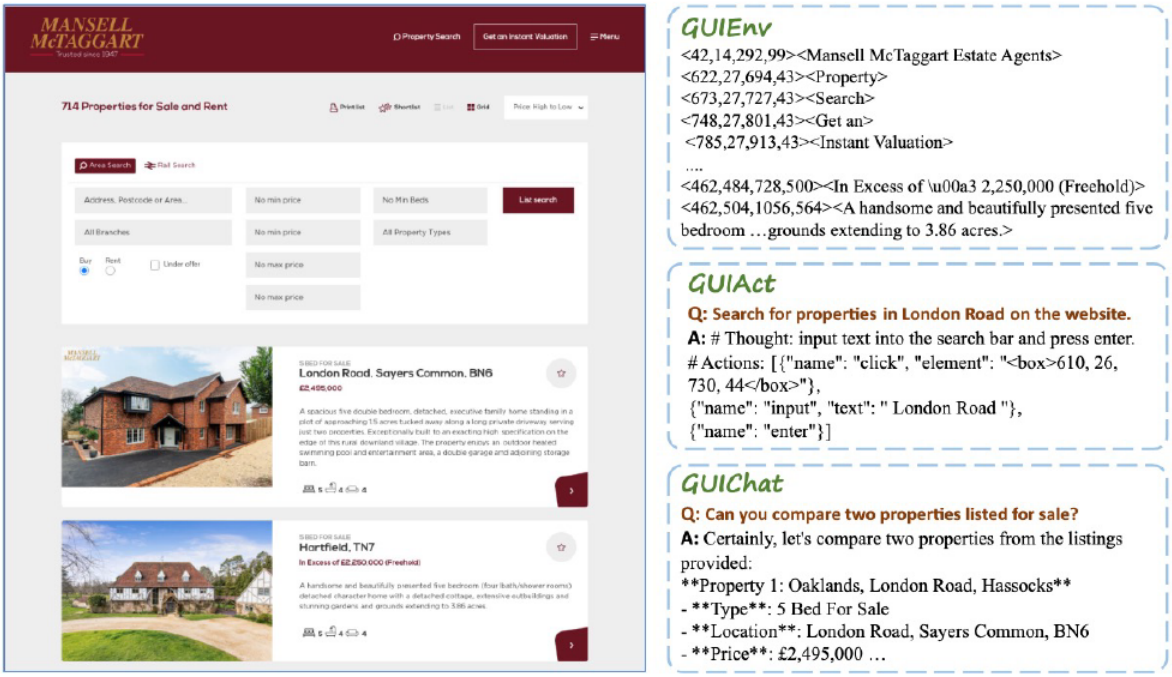
WebShop

agent学习网上购物



GUIAgent

学习操作GUI工具，与VLM模型结合



教程学习：让模型阅读工具手册来学习

OpenAI的大模型都有很强的zero-shot能力，能够理解手册的内容

Zero-shot & Few-shot 的例子

Zero-shot Prompting: Here we provide a tool (API) "forecast_weather(city:str, N:int)", which could forecast the weather about a city on a specific date (after N days from today). The returned information covers "temperature", "wind", and "precipitation".
Please write codes using this tool to answer the following question: "What's the average temperature in Beijing next week?"

Few-shot Prompting: We provide some examples for using a tool. Here is a tool for you to answer question:
Question: "What's the temperature in Shanghai tomorrow?"

```
return forecast_weather("Shanghai", 1)["temperature"]
```

Question: "Will it rain in London in next two days?"

```
for i in range(2):
    if forecast_weather("London", i+1)["precipitation"] > 0:
        return True
return False
```

Question: "What's the average temperature in San Francisco next week?"

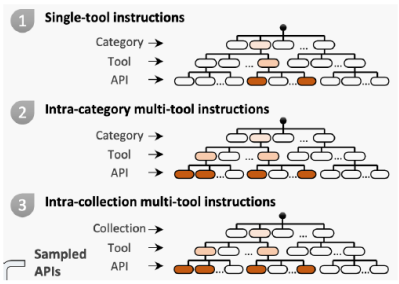
ToolBench

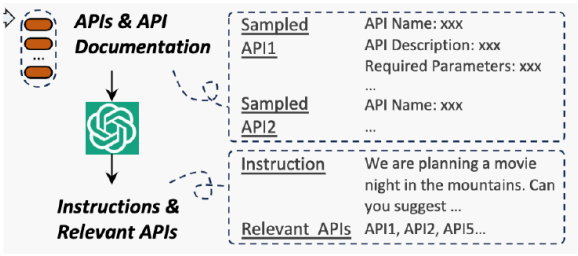
https://github.com/OpenBMB/ToolBench

- 集成了来自RapidAPI超过16000个API
  - 选取了16, 000多个高质量API
  - 涵盖了49个类别
- 支持单工具或多工具的调用
  - 简单的api指令集合
  - chatgpt自动生成指令，可能包括一个或多个api

Instruction Generation

- Single Tool + Multi-Tool
- (1) Sample a collection of APIs: $\mathbb{S}_N^{sub} = \{API_1, \cdots, API_N\}$
- (2) ChatGPT automatically generate instructions that may require calling one or more APIs in the collection: $\underset{\{API_1,\cdots,API_N\}\in\mathbb{S}_{API},\ \{seed_1,\cdots,seed_3\}\in\mathbb{S}_{seed}}{\text{ChatGPT}}(\{[\mathbb{S}_1^{rel}, Inst_1], \cdots, [\mathbb{S}_{N'}^{rel}, Inst_{N'}]\}|API_1, \cdots, API_N, seed_1, \cdots, seed_3).$





- 支持复杂的推理任务

Resource	ToolBench (this work)	APIBench (Patil et al., 2023)	API-Bank (Li et al., 2023a)	ToolAlpaca (Tang et al., 2023)	T-Bench (Xu et al., 2023b)
Real-world API?	✓	✗	✓	✗	✓
Real API Response?	✓	✗	✓	✗	✓
Multi-tool Scenario?	✓	✗	✗	✗	✗
API Retrieval?	✓	✓	✗	✗	✗
Multi-step Reasoning?	✓	✗	✓	✓	✓
Number of tools	**3451**	3	53	400	8
Number of APIs	**16464**	1645	53	400	232
Number of Instances	12657	**17002**	274	3938	2746
Number of Real API Calls	**37204**	0	568	0	0
Avg. Reasoning Traces	4.1	1.0	2.1	1.0	**5.9**

一个模型学习使用工具的例子：VPT

Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos NeurIPS 2022

Memory: manage the working history

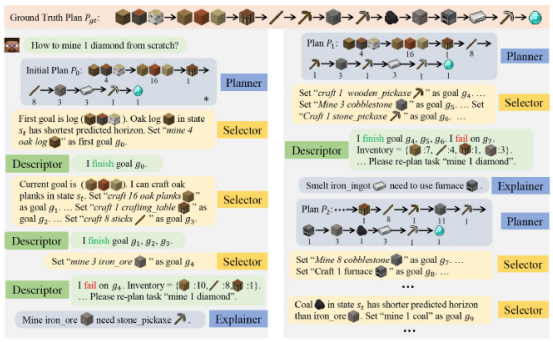
Short-Term Memory 短期记忆

短时记忆通常是通过上下文学习实现的，记忆信息直接写入prompt中

No external memory storage

```
# RLP_gpt4

Initialize {
   My personality is [[PERSONALITY]]
}

Think {
   This last message made me feel ...
   My previous plan was ...
   I think ...
   I will send the message, " ...
   In retrospect ...
   My next plan is ...

   constraints {
       Output format in squiggly brackets separated by newlines
       Only put quotes surrounding the message
   }
}

Execute Think(new message)
```

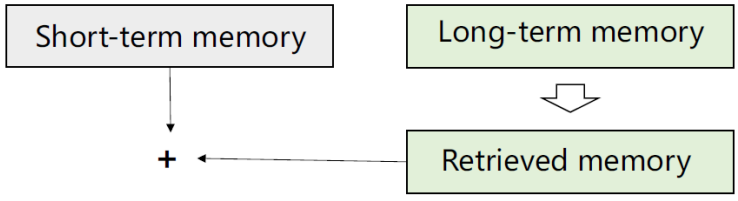
Reflective Linguistic Programming (RLP): A Stepping Stone in Socially-Aware AGI (SocialAGI)

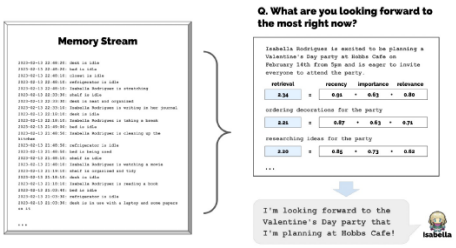


Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents

Short-Term Memory + Long-Term Memory 短期+长期记忆

外部记忆存储+检索外部记忆+短期记忆





Short-term memory

current state, agent profile, ...

Long-term memory

Retrieved information from the memory stream

如何存储长期记忆

直接写入Raw Text

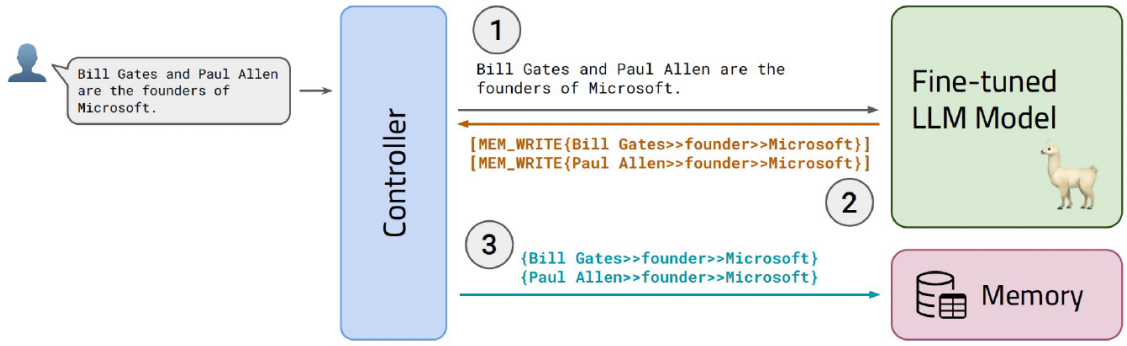
Raw Text ⇨ Memory Write

Memory Stream

```
2023-02-13 22:48:20: desk is idle
2023-02-13 22:48:20: bed is idle
2023-02-13 22:48:10: closet is idle
2023-02-13 22:48:10: refrigerator is idle
2023-02-13 22:48:10: Isabella Rodriguez is stretching
2023-02-13 22:33:30: shelf is idle
2023-02-13 22:33:30: desk is neat and organized
2023-02-13 22:33:10: Isabella Rodriguez is writing in her journal
2023-02-13 22:18:10: desk is idle
2023-02-13 22:18:10: Isabella Rodriguez is taking a break
2023-02-13 21:49:00: bed is idle
2023-02-13 21:48:50: Isabella Rodriguez is cleaning up the
kitchen
2023-02-13 21:48:50: refrigerator is idle
2023-02-13 21:48:50: bed is being used
2023-02-13 21:48:10: shelf is idle
2023-02-13 21:48:10: Isabella Rodriguez is watching a movie
2023-02-13 21:19:10: shelf is organized and tidy
2023-02-13 21:18:10: desk is idle
2023-02-13 21:18:10: Isabella Rodriguez is reading a book
2023-02-13 21:03:40: bed is idle
2023-02-13 21:03:30: refrigerator is idle
2023-02-13 21:03:30: desk is in use with a laptop and some papers
on it

...
```

编码后写入

Raw Text ⇨ Symbolic ⇨ Memory Write

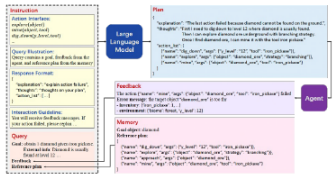


(a) Memory-Write scenario: (1) Controller passes the input to the LLM (2) which generates the appropiate memory write call. (3) The controller gives the data (and their average represntations) to the memory to be stored.

存储策略

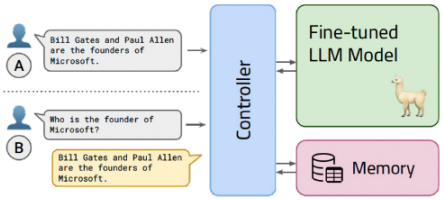
- 基于LLM合并相同的记忆
- 遵循先进先出的原则，最早的记忆会首先被覆盖

Memory Duplicated



- Agent in Minecraft
- Store success action sequence for each task
- Merge similar memory based on LLM

Memory Overflow

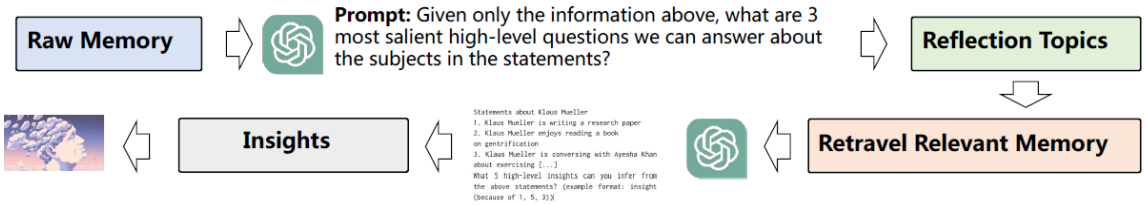


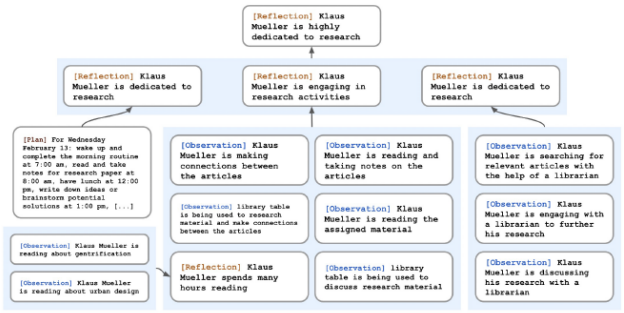
First-in-first-out (FIFO)

earlier memories are more likely to be removed.

人类对记忆能够自证和评估，我们希望模型也可以

Self-summarization



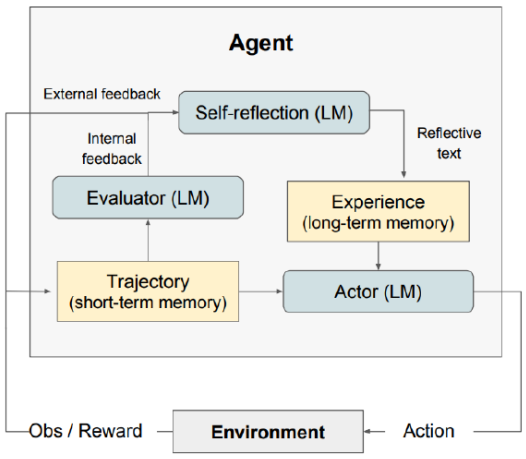


Self-verification

Inventory (8/36): {'oak_planks': 5, 'cobblestone': 2, 'porkchop': 2, 'wooden_sword': 1, 'coal': 5, 'wooden_pickaxe': 1, 'oak_log': 3, 'dirt': 9} **Task**: Mine 5 coal ores	GPT-4	**Reasoning**: Mining coal_ore in Minecraft will get coal. You have 5 coal in your inventory. **Success**: True
Inventory (10/36): {'raw_copper': 9, 'copper_ingot': 3, 'acacia_planks': 1, 'raw_iron': 1, 'stick': 1, 'iron_sword': 1, 'iron_pickaxe': 1, 'iron_ingot': 3, 'crafting_table': 1, 'furnace': 1} **Task**: Craft a spyglass	GPT-4	**Reasoning**: To craft a spyglass, you need 2 copper ingots and 1 amethyst shard. You have 3 copper ingots, but you don't have any amethyst shards. **Success**: False **Critque**: Find and mine an amethyst shard underground.
Inventory (7/36): {'oak_log': 5, 'oak_planks': 1, 'wooden_pickaxe': 1, 'wooden_sword': 1, 'porkchop': 2, 'white_wool': 2, 'mutton': 6} **Task**: Kill 3 sheep	GPT-4	**Reasoning**: You have 2 white_wool and 6 mutton in your inventory, which indicates that you killed 2 sheep. You needed to kill 3 sheep. **Success**: False **Critique**: Find and kill one more sheep to complete the task.
Inventory (9/36): {'string': 4, 'coal': 1, 'rotten_flesh': 1, 'iron_sword': 1, 'furnace': 1, 'dirt': 6, 'stone_shovel': 1, 'wooden_pickaxe': 1, 'granite': 5} **Task**: Kill 1 zombie	GPT-4	**Reasoning**: You have 1 rotten_flesh in your inventory, which means you have killed at least 1 zombie. **Success**: True

Reflexion NeurIPS 2023

利用语言反馈信号强化agent，从之前的失败中吸取教训



Algorithm 1 Reinforcement via self-reflection

Initialize Actor, Evaluator, Self-Reflection: M_a, M_e, M_{sr}
Initialize policy $\pi_\theta(a_i|s_i), \theta = \{M_a, mem\}$
Generate initial trajectory using π_θ
Evaluate τ_0 using M_e
Generate initial self-reflection sr_0 using M_{sr}
Set $mem \leftarrow [sr_0]$
Set $t = 0$
while M_e not pass or $t <$ max trials **do**
  Generate $\tau_t = [a_0, o_0, \dots a_i, o_i]$ using π_θ
  Evaluate τ_t using M_e
  Generate self-reflection sr_t using M_{sr}
  Append sr_t to mem
  Increment t
end while
return

Agent的安全性讨论

- Agent本身可能会被注入目的性的引导，例如用来购物的agent可能会被操控倾向于选择某类商品，这对用户是很难以察觉的
- Agent调用的API本身安全性是否能够得到保障？有些工具本身就存在一些不安全的因素
- Agent调用某个工具的原因是一个黑盒，这对一些敏感场景有风险（自动驾驶 医疗系统）

Report 7. 工业界专场

百川 技术负责人 方琨

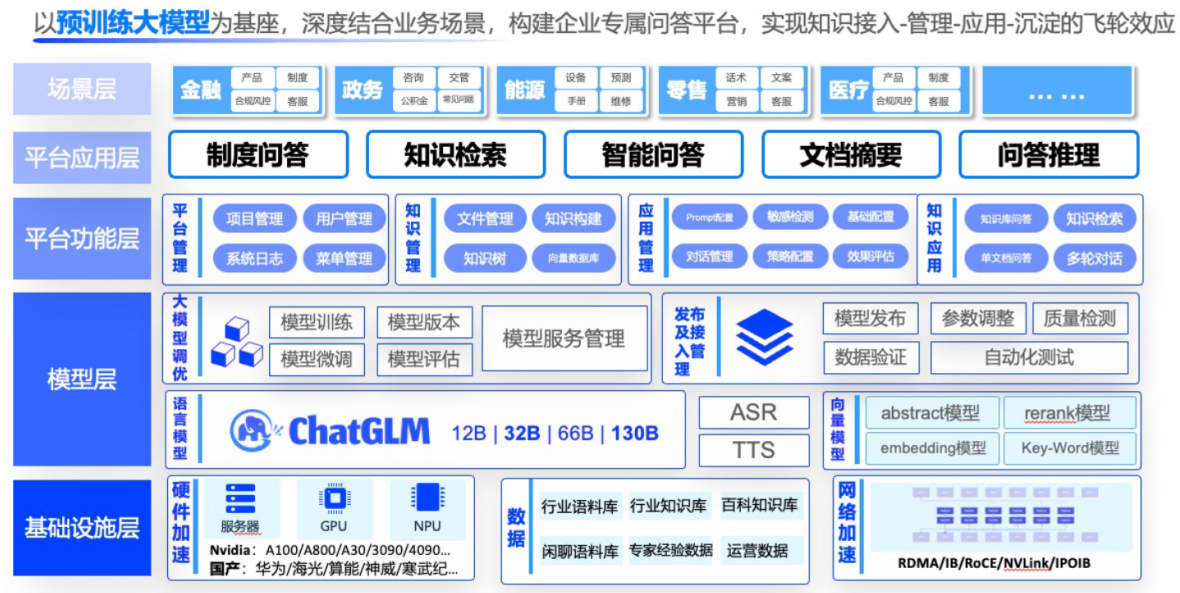
智谱 解决方案专家 冯小平

Jina AI 联合创始人兼CEO 王楠

主要观点

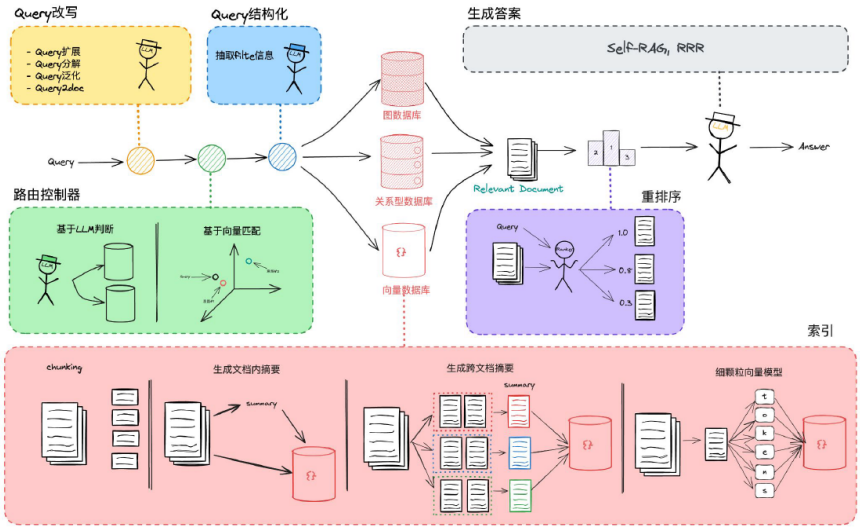
- 意图理解
  - 更多的去解读用户意图，适应用户意图
  - 针对特定的应用场景，将大模型从通用→专用
- 大规模处理
  - 近期GPT4出现连接云盘的接口，目前只能处理上传一个文件
  - 但这是一个未来趋势，能够整合云盘中大量的结构化、非结构化数据
- 企业的tool calling就绪度目前还差很多
- 很多时候更加专注延迟等用户体验的指标

企业大模型的一个简单框架



一个很好的样例：

- Query改写
- 路由控制器
- Query结构化
- 索引优化
- 重排序
- ...

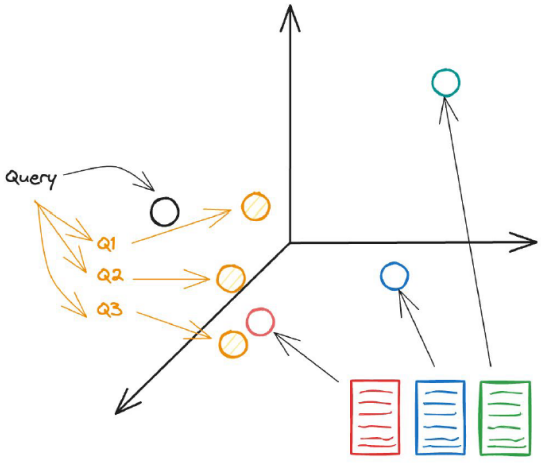


Query改写

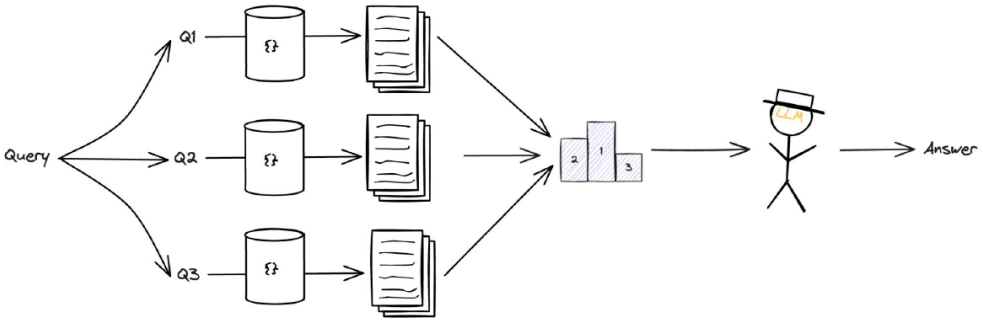
使Query和Document更容易匹配

举例："怎么添加wx支付信息?"

- 等价的Query
  - "怎么添加微信支付信息?"
- 更抽象的Query
  - "如何补充支付信息？"
- 更具体的Query
  - "如何找到用户设置页面？"
  - "如何在用户设置页面中补充支付信息？"
  - "如何选择微信支付？"



等价Query：并行多查询



- 使用LLM生成等价query
- 多路并行召回
- 使用排序模型或启发式合并

Query: What is the BLEU score of transformer on WMT 2014?

- Q1: Can you provide the BLEU score achieved by the transformer model on the WMT 2014 dataset?
- Q2: What was the BLEU score obtained by the transformer architecture when evaluated on the WMT 2014 dataset?
- Q3: How well did the transformer model perform in terms of BLEU score on the WMT 2014 dataset?

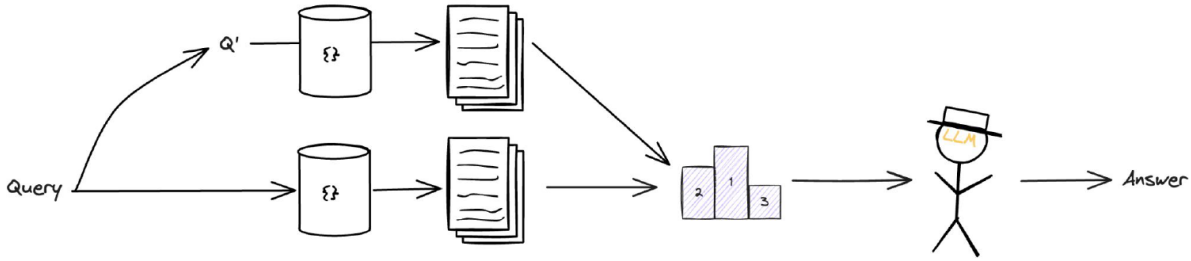
```
The BLEU score of the
Transformer on WMT 2014 is
28.4, establishing a new
state-of-the-art score.
Additionally, the big model
achieves a BLEU score of 41.0
on the English-to-French
translation task
```

```
The BLEU score of the
Transformer on the WMT 2014
English-to-German translation
task is 28.4, and on the WMT
2014 English-to-French
translation task, it is 41.8.
```

更抽象的Query：回撤

更抽象的Query：回撤

- 使用LLM对问题进行抽象
- 使用原始Query和改写后Query并行召回
- 适用于处理细颗粒的具体问题



Query: How many GPUs are needed for training transformer?

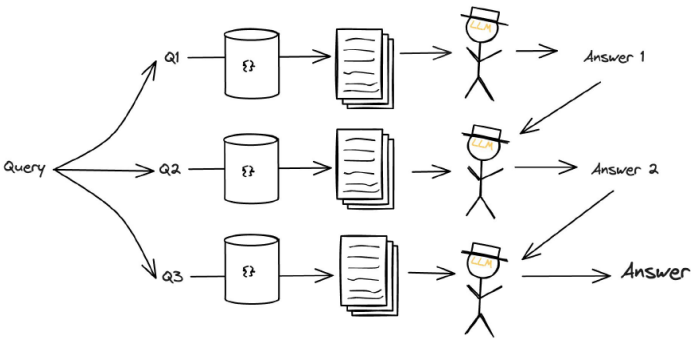
Q': what hardware is required for training transformer?'

Based on the provided context, the recommended number of GPUs for training transformer models efficiently is eight. The Transformer model achieved a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, which was a small fraction of the training costs of the best models from the literature.

To train the Transformer model, a total of eight GPUs are needed. This is based on the information provided in the documents, where it states that the models were trained on one machine with 8 NVIDIA P100 GPUs. The training process involved both base models and big models, with different hyperparameters and training steps. The big models, described at the bottom line of table 3, were trained for 300,000 steps over 3.5 days using the eight GPUs. This setup allowed for efficient training of the Transformer model, showcasing its ability to achieve state-of-the-art results in machine translation tasks while being more parallelizable and requiring significantly less time to train compared to other models based on recurrent or convolutional neural networks.

更具体的Query：任务分解

- 使用LLM拆解问题
- 根据上一步问题的答案和当前的问题生成答案
- 多路串行召回
- 适合语义颗粒度大的抽象问题



Query: What are encoder and decoder stacks in transformer?

Q1. How do encoder stacks work in a transformer model?

Q2. What is the purpose of decoder stacks in a transformer architecture?

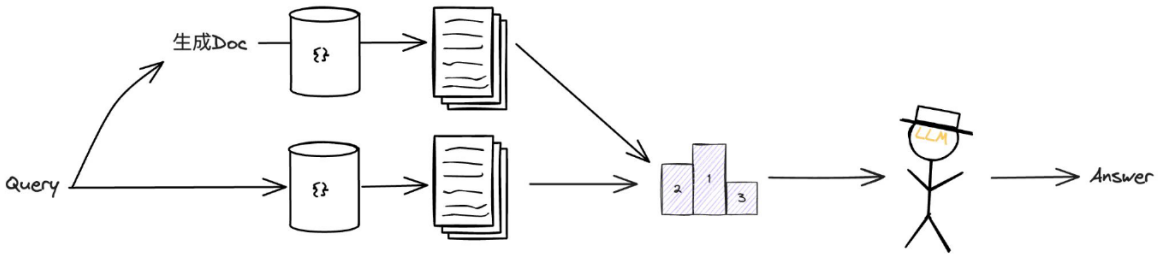
Q3. Can you explain the differences between encoder and decoder stacks in a transformer?

In the Transformer model, the encoder stack is composed of a stack of N=6 identical layers, each consisting of two sub-layers - a multi-head self-attention mechanism and a simple, position-wise fully connected feed-forward network. The decoder stack is also composed of a stack of N=6 identical layers, with each layer containing two sub-layers from the encoder stack and an additional third sub-layer for multi-head aligned RNNs or convolution.

In a transformer model, the encoder stack is composed of N = 6 identical layers, each containing two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. …
On the other hand, the decoder stack in a transformer also consists of N = 6 identical layers. … the decoder stack is responsible for producing the desired output sequence based on the continuous representations provided by the encoder.

根据Query生成Document：HyDE

使用查询到的文档+Query进行查询



- Document有更丰富的信息，可以更精准的匹配
- 用LLM根据Query生成Document
- 使用原始Query和生成的Document并行召回

Query: What learning rate is used when training transformer?

The learning rate used when training a transformer model is a crucial hyperparameter that significantly impacts the model's performance and convergence speed. In practice, a common learning rate schedule for training transformers involves using a warmup strategy followed by a decay schedule. During the warmup phase, the learning rate is gradually increased from a small value to the specified initial learning rate over a certain number of steps. This helps the model to stabilize and avoid large fluctuations in the early stages of training. \n\nAfter the warmup phase, the learning rate is typically decayed exponentially or according to a predefined schedule to fine-tune the model's parameters and improve convergence. The specific learning rate values and decay schedule can vary depending on the dataset, model architecture, and training objectives. ...

The learning rate used when training the transformer is warmup_steps = 4000, decreasing proportionally to the inverse square root of the step number. The base model of the transformer uses a training cost of 3.3 · 10^18 FLOPs, while the big model uses a training cost of 2.3 · 10^19 FLOPs. The base model applies a dropout rate of Pdrop = 0.1 during training.

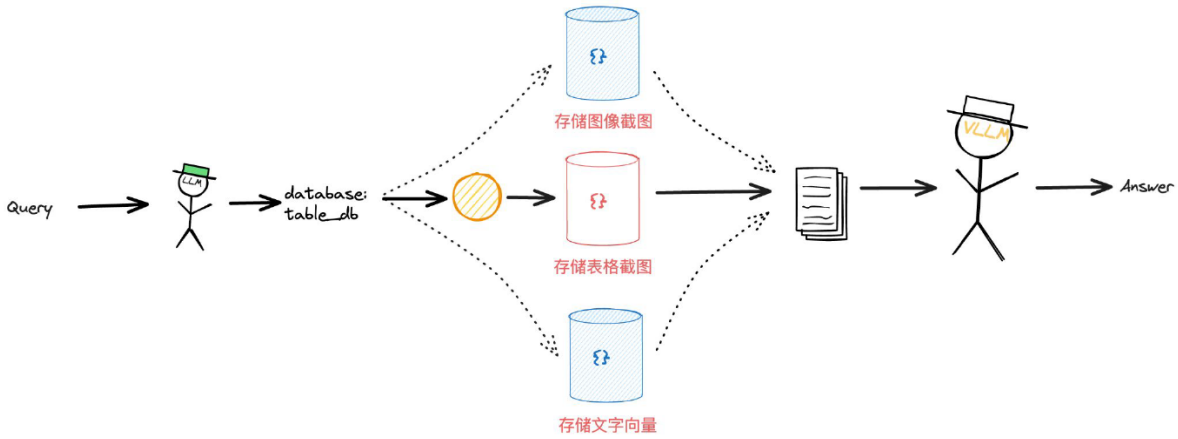
The learning rate used when training the Transformer is determined by the formula:

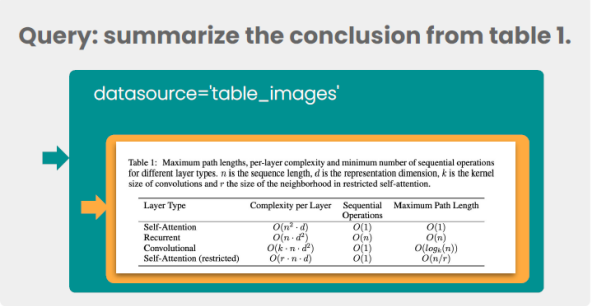
lrate = d^-0.5 * model * min(step_num^-0.5, step_num * warmup_steps^-1.5)

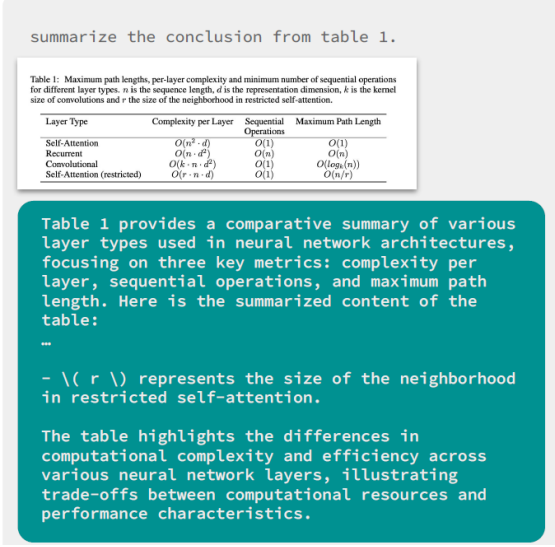
This formula increases the learning rate linearly for the first warmup_steps training steps and then decreases it proportionally to the inverse square root of the step number. The warmup_steps value used in this case is 4000.

路由控制器

- 意图识别
- 根据用户意图决定召回路径







Jina AI的RAG工具

jina-embeddings-v2

23.10发布，全球第一个支持8k输入长度的开源向量模型

jina-colbert-v1

第一款支持8k长度的colbert模型

jina-reranker-v1

基于jina bert v2 支持8k上下文输入

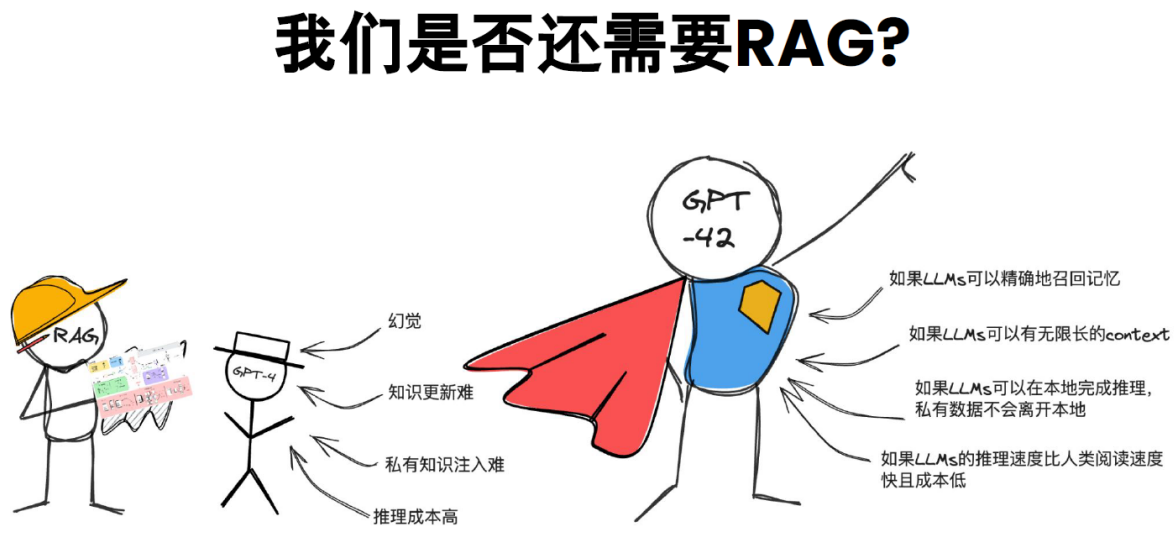
jina-clip-v1

正在开发...

AIR-Bench

自动化多样信息检索评测基准

思考和讨论



如果未来LLM能够精确的召回记忆，那么RAG将不再被需要。