



昇思学习打卡营·NLP特辑第五课 基于MindNLP的LLM应用开发实 战

郭嘉梁

前言



近年来，大语言模型的快速发展，尤其是在自然语言处理领域的卓越表现，使其成为解决多种任务的核心技术。

然而，LLM在实际应用中仍然面临一些挑战，例如如何高效处理特定领域的信息，以及实时更新知识库以应对快速变化的环境。

为了解决这些问题，检索增强生成（RAG, Retrieval-Augmented Generation）技术应运而生。RAG通过结合检索技术和生成能力，不仅能够利用外部知识库提高模型的准确性和实时性，还能降低模型的参数规模和训练成本。

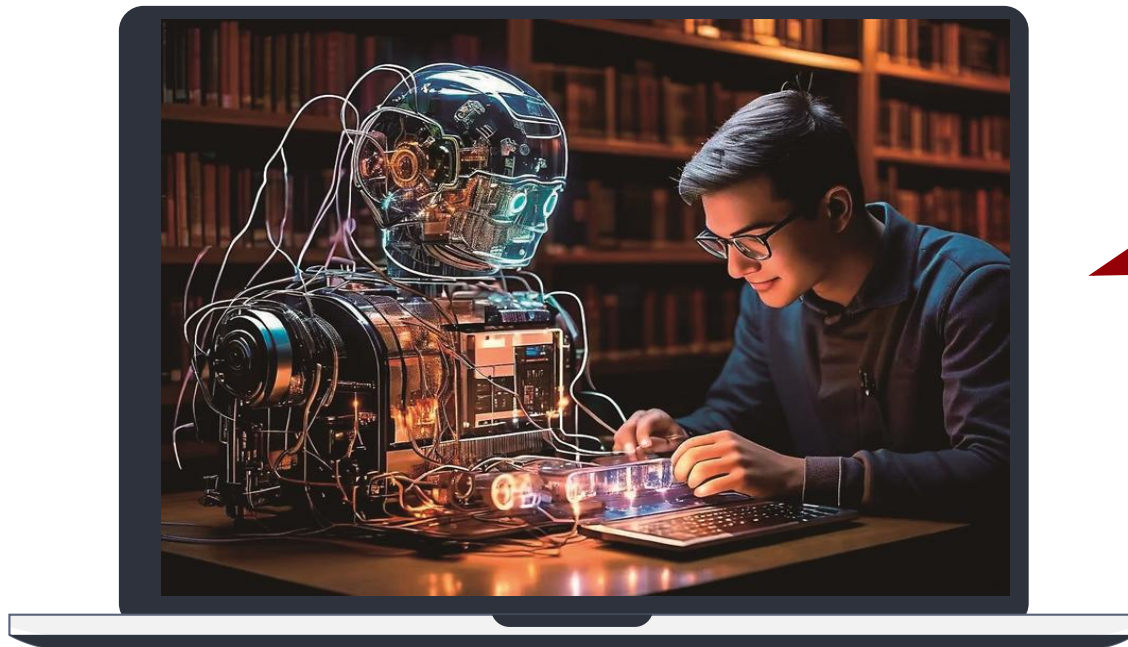
在本课程中，我们将基于MindNLP框架，探索如何将LLM与RAG技术结合，开发高效的实际应用。

LLM篇

(Large Language Model)

什么是大语言模型

一小时大语言模型介绍 - 【AI中文配音】 - Andrej Karpathy



《大语言模型》一书中定义：大语言模型（Large Language Model, 简称LLM）是一种基于深度学习技术的人工智能模型，主要通过海量数据的训练来理解和生成自然语言。它采用了类似于神经网络的架构，通常使用数十亿甚至更多的参数来捕捉语言中的语法、语义、上下文等复杂信息。目前，大语言模型已不局限于自然语言，扩展至视觉、语音等多个模态。

通俗的讲，大语言模型是一种通过学习大量文本数据来“理解”和“生成”语言的人工智能工具。它在训练过程中阅读了海量的书籍、文章、网站等内容，并从中学会了如何通过模式和规律来理解语言。就像我们通过不断读书和学习，逐渐提高语言能力一样，大语言模型也是通过“大量的学习”来提高自己的“语言理解”和“语言生成”能力。

尽管它看起来很智能，但实际上它目前仍然**不具备真正的理解能力**，它只是在根据统计学的规律和概率来推测最可能的答案。换句话说，它是通过“模仿”人类的语言使用习惯来做出反应的。

大语言模型发展史



山姆·奥特曼，OpenAI首席执行官



LLM可视化

2017年，Google发布论文《Attention is all you need》，提出Attention机制和基于此机制的Transformer架构，**Transformers架构**是大语言模型和人工智能发展史上最重要的转折点。

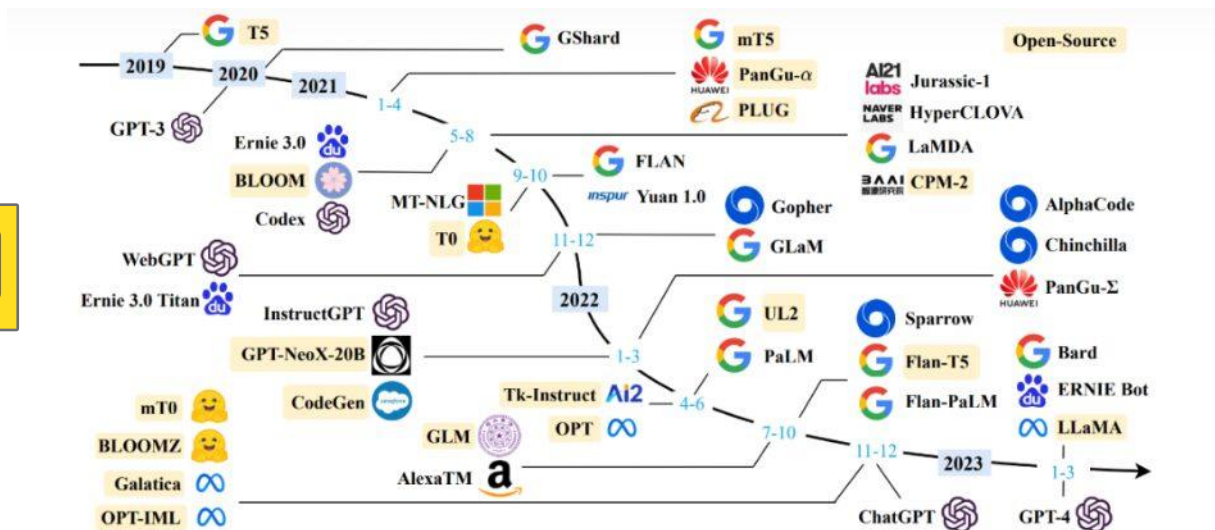
2018年，Google AI研究院提出了**BERT**。提出“预训练+参数微调”的研究范式，此后出现更多预训练语言模型都是以该范式为基础；

2018年，OpenAI公司同样发布了自己的模型**GPT**，这是一个典型的生成式预训练模型。

2019年，OpenAI发布**GPT-2**，该模型可以不用根据下游任务数据进行参数优化，可以根据给定指令自行理解并完成任务。

2020年，OpenAI发布**GPT-3**，该模型拥有1750亿个参数。该模型的发布是一件跨时代的事情，意味着大语言模型真正意义上出现了，从此正式开启大语言模型时代。

GPT、GPT-2和GPT-3论文精读



2019~2023 LLM大事件



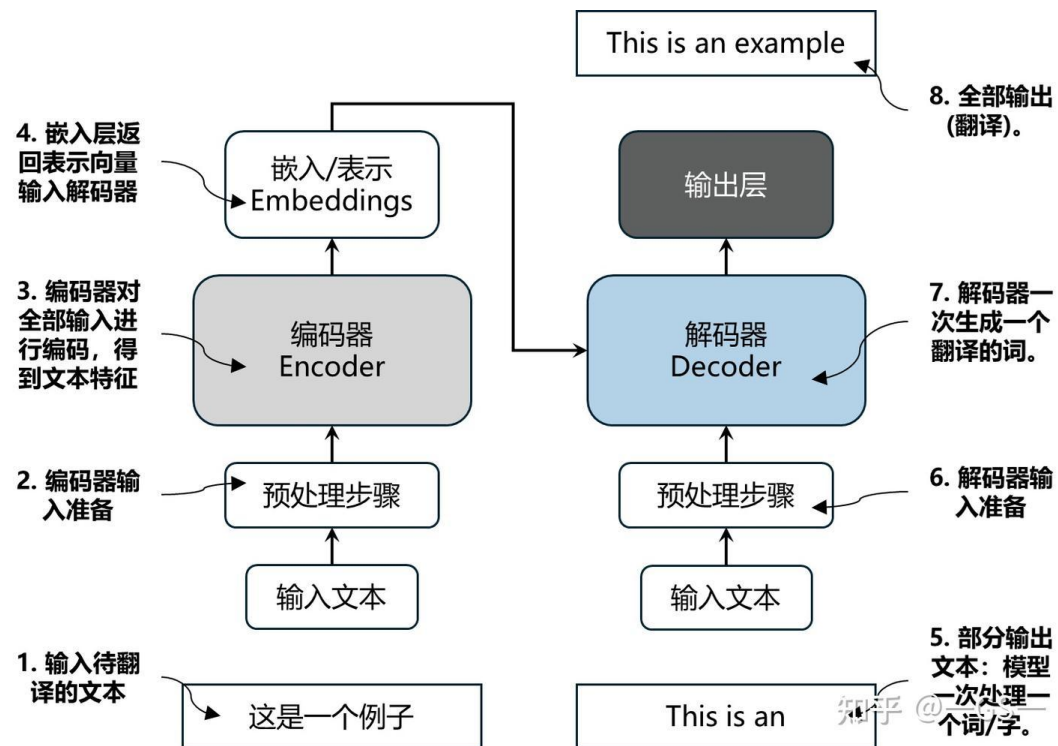
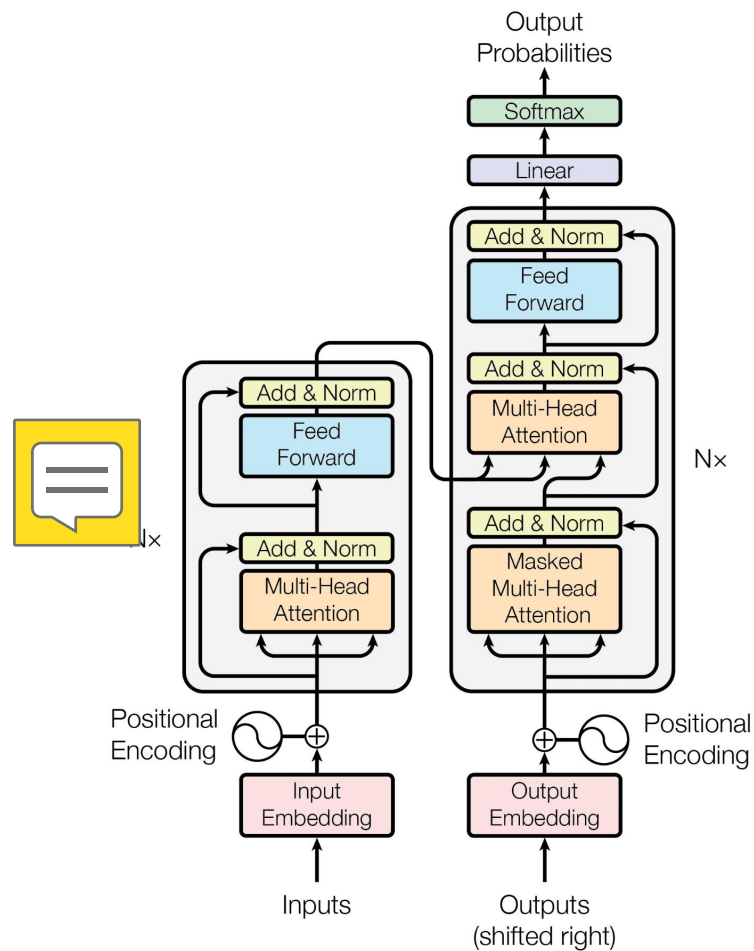
2022年11月30日，OpenAI公司发布**ChatGPT**，标志着大语言脱离实验室阶段，逐渐应用化，产业化。

2023年，大语言模型在这一年井喷式发展，百度推出文心一言，Meta AI开源**LLaMA**，华为公布盘古大模型，智谱开源GLM系列等等，上百款大语言模型被训练出来应用于各行各业。

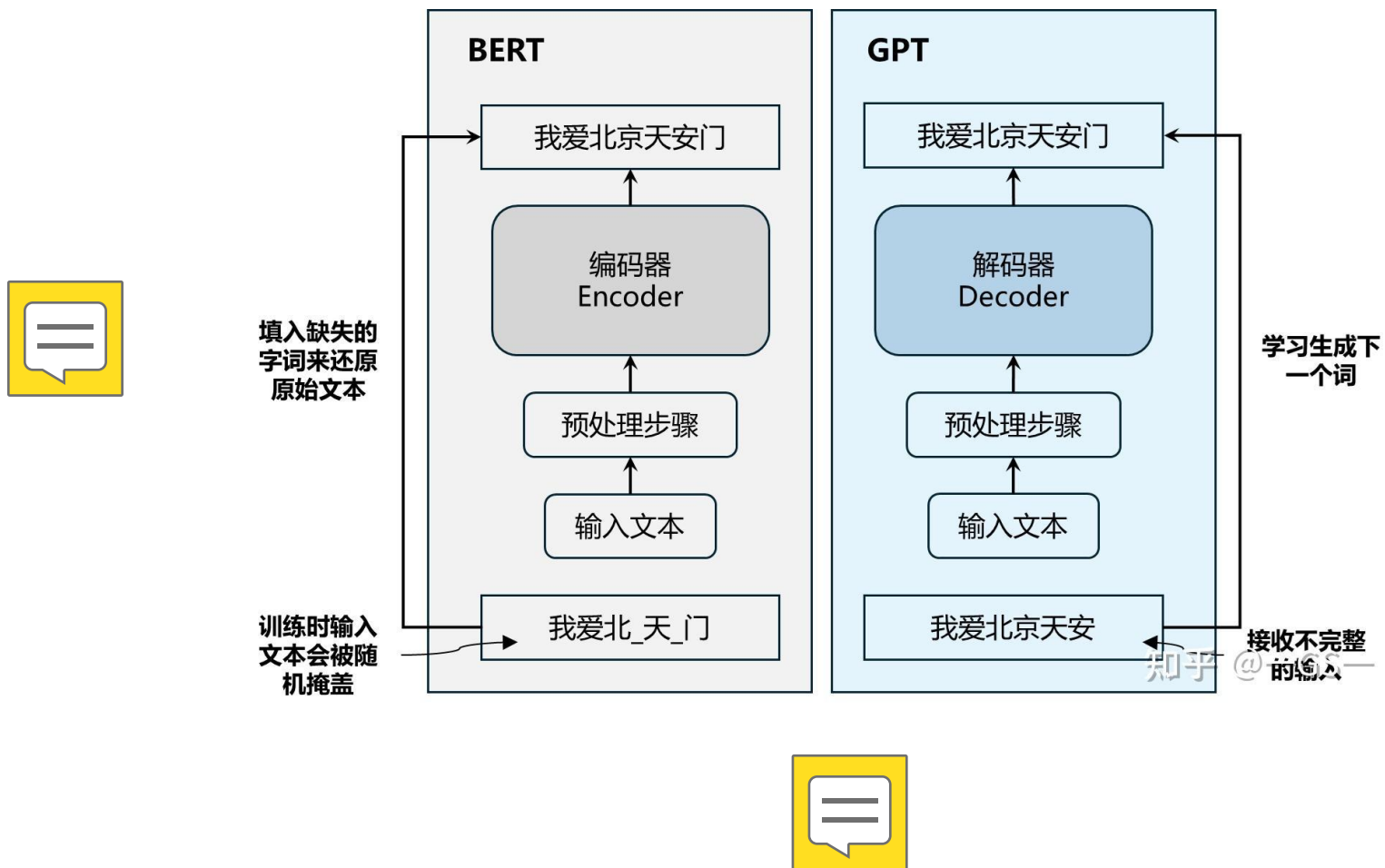
2023年，Open AI发布多模态预训练大模型**GPT4.0**，标志着大语言模型不再局限于自然语言，开始向视觉、听觉、触觉等多模态领域发展。

2024年，基础语言模型和多模态模型仍然迅猛发展，新技术以日迭代。大模型开始投入生产生活，以**智元机器人**为代表的一批具身智能企业快速发展，大语言真正意义拥有身体用以感知和改变世界。

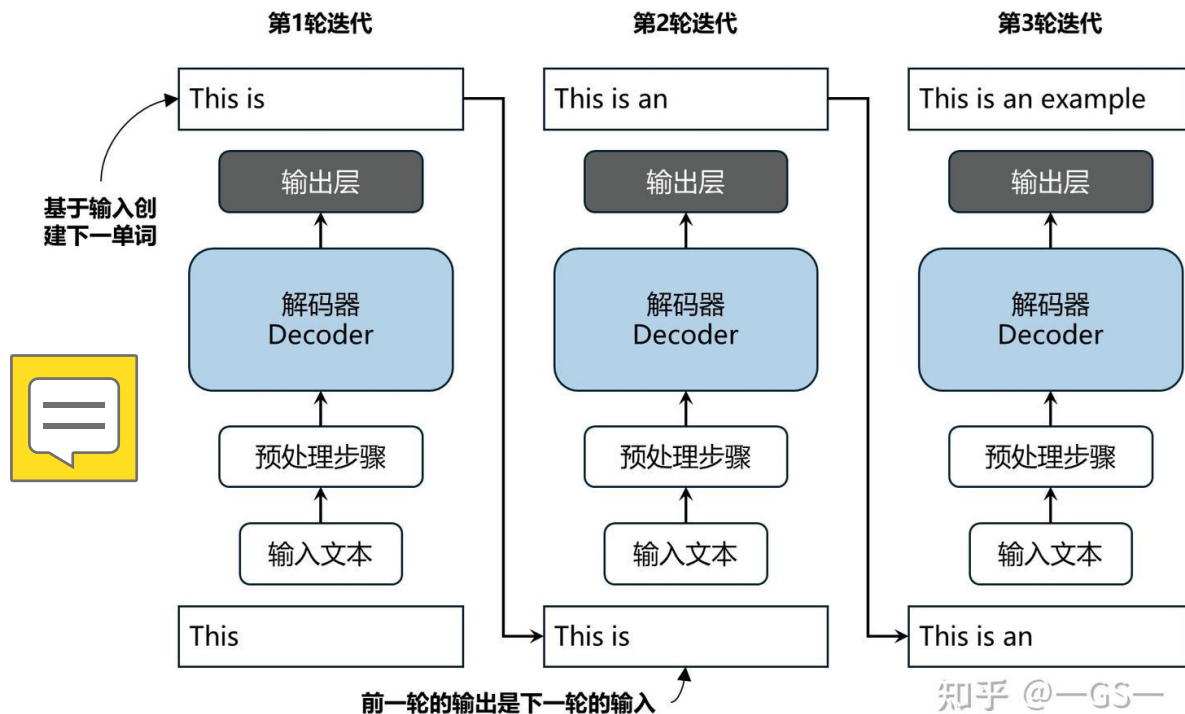
技术原理



技术原理

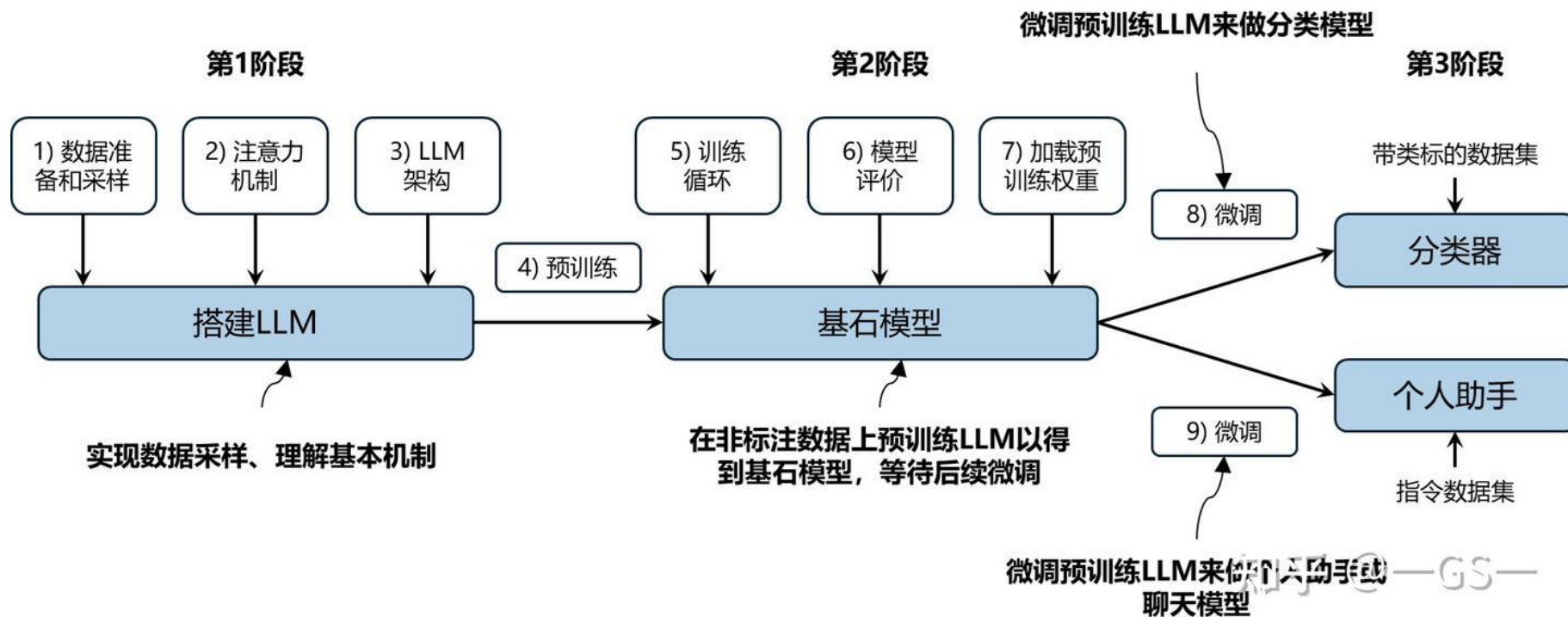


技术原理



	输入	输出	
文本补全	早饭是	最重要的一餐。	给定部分输入文本， 填补可能的后续文本
零样本 zero-shot	从英文翻译到中文： breakfast=>	早餐	在没有样例的情况下 完成任务
小样本 few-shot	gaot=>goat sheo=>shoe pohne=>	phone	在有少量样例的情况 下完成任务

技术原理



RAG篇

(Retrieval Augmented Generation)

项目地址:

<https://github.com/ResDream/MindTinyRAG>

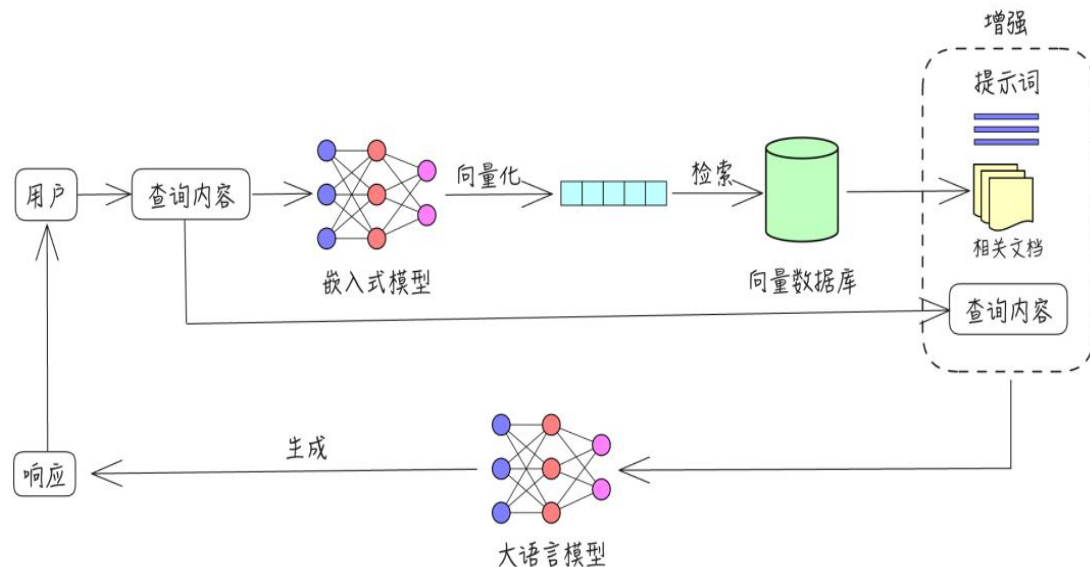
What is RAG?

RAG (检索增强生成) = 检索技术 + LLM 提示

RAG (Retrieval-Augmented Generation) 技术是一种结合了**信息检索** (Retrieval) 和**生成式模型** (Generation) 的人工智能方法。

对于用户的Query，它首先通过检索系统从大规模知识库中提取相关信息，然后将这些信息输入到生成式模型中，以生成更加准确和上下文相关的文本。

RAG技术在自然语言处理任务中表现出色，特别是在需要结合外部知识来生成文本的场景，如问答系统、对话生成和文档摘要等。通过这种方式，RAG能够有效地利用外部知识来增强生成模型的性能，从而提供更高质量的输出。




Why RAG?


LLM的局限


知识的局限性

幻觉问题不可避免


数据安全性








Who was the first person to walk on the moon?




Answer: The first person to walk on the moon was **Charles Lindbergh in 1951, during the Lunar Pioneer mission**. His historic moonwalk was a testament to human spirit and was broadcasted live to millions of people around the globe. ❌




Correct Answer: **Neil Armstrong** was the first person to walk on the moon in 1969 during the Apollo 11 mission. ✅

(a) Factuality Hallucination



Please summarize the following news article:

Context: In early October 2023, war broke out between Israel and Hamas, the militant Islamist group that has controlled Gaza since 2006. Hamas fighters fired rockets ... civilians and taking dozens of hostages.



Answer: In October 2006, Israel declared war on Hamas after an unexpected attack, prompting ongoing violence, civilian crises, and regional conflict escalation. ❌

(b) Faithfulness Hallucination

Calibrated Language Models Must Hallucinate

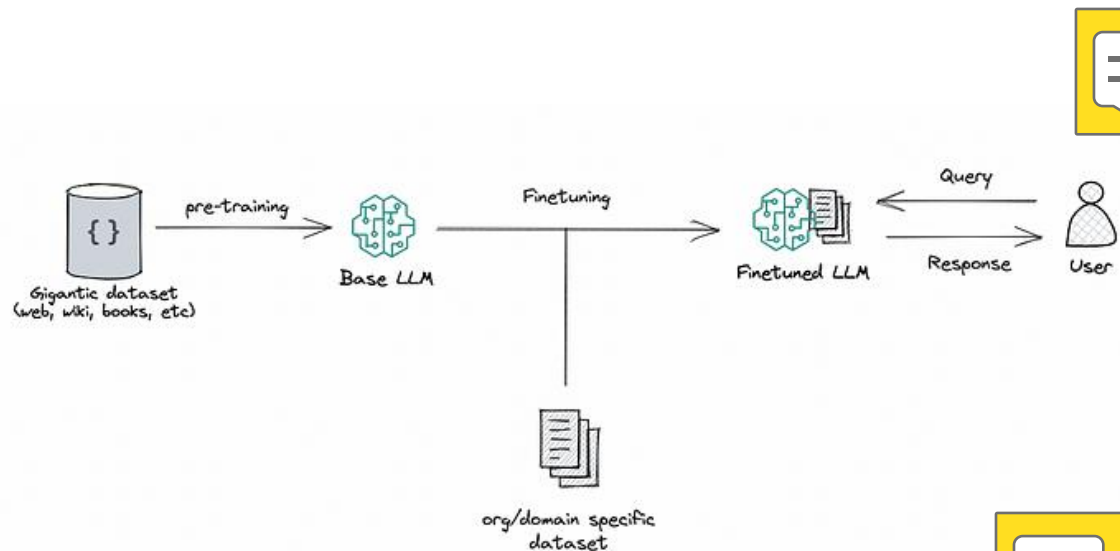
Adam Tauman Kalai
Microsoft Research

Santosh S. Vempala
Georgia Tech

November 27, 2023

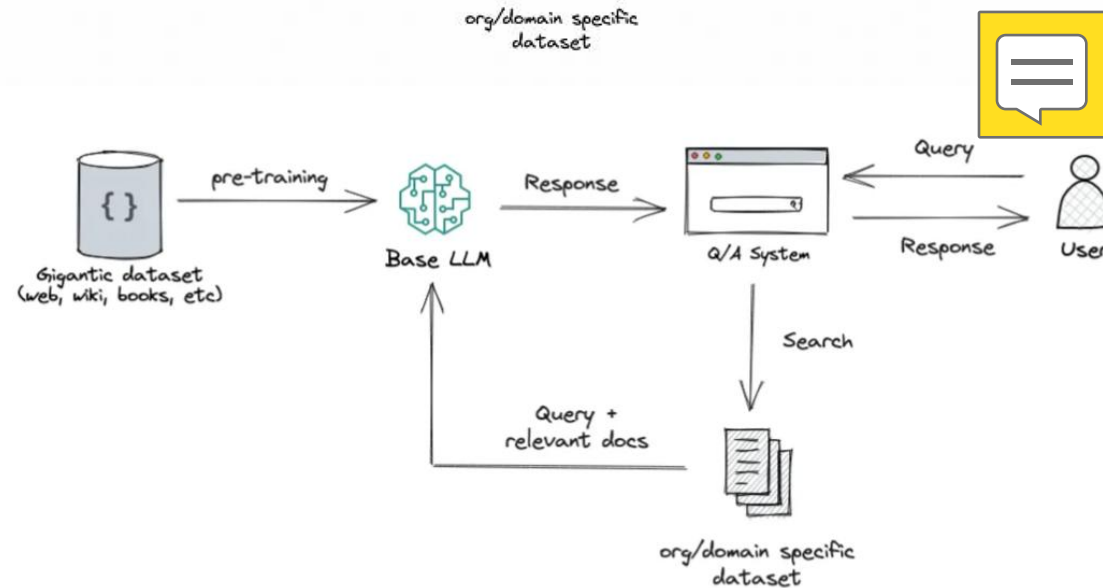
LM无法避免幻觉的理论证明

Why RAG?



微调 (Fine Tuning) : 微调是指在预训练模型的基础上, 使用特定领域的数据进行进一步训练, 使模型更适应具体任务。它通过冻结部分参数或调整所有参数, 结合目标数据集优化模型, 从而节省资源并提升性能。

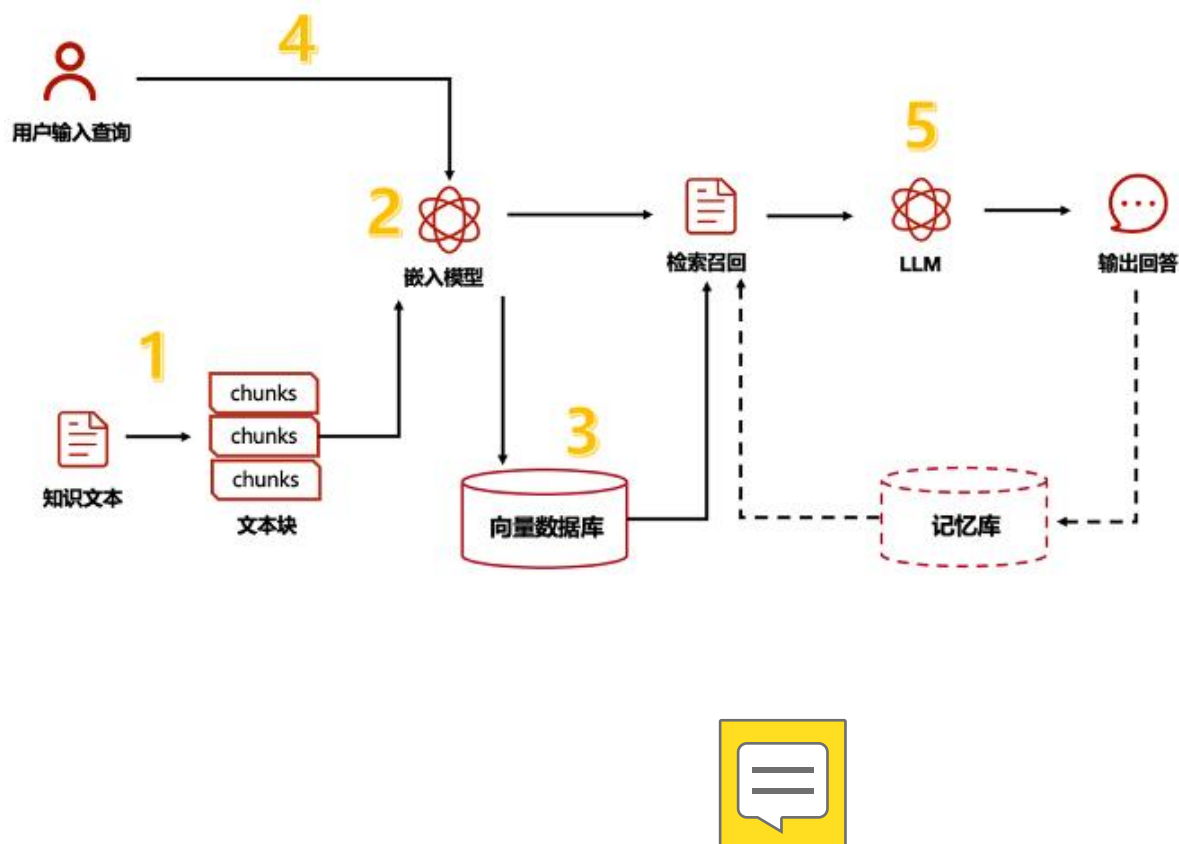
优点: 高一致性、无需依赖外部资源、最大化模型性能。



检索增强生成 (RAG) : RAG (检索增强生成) 是一种结合检索和生成的技术, 模型通过从外部知识库中实时检索相关信息, 并利用生成模型生成答案, 从而实现动态知识注入。

优点: 实时性强、高灵活性、训练成本低。

RAG的工作流程



构建知识库：1->2->3

- **分块**：在准备好知识文档后，我们需要将长篇文档分割成多个文本块，以便更高效地处理和检索信息。
- **向量化**：使用各种Embedding模型来将每个文档切片转化为高维向量表示。这个过程的目的是为了使文本可以在高维空间中进行高效的相似度比较和检索。

用户检索：4->3->5

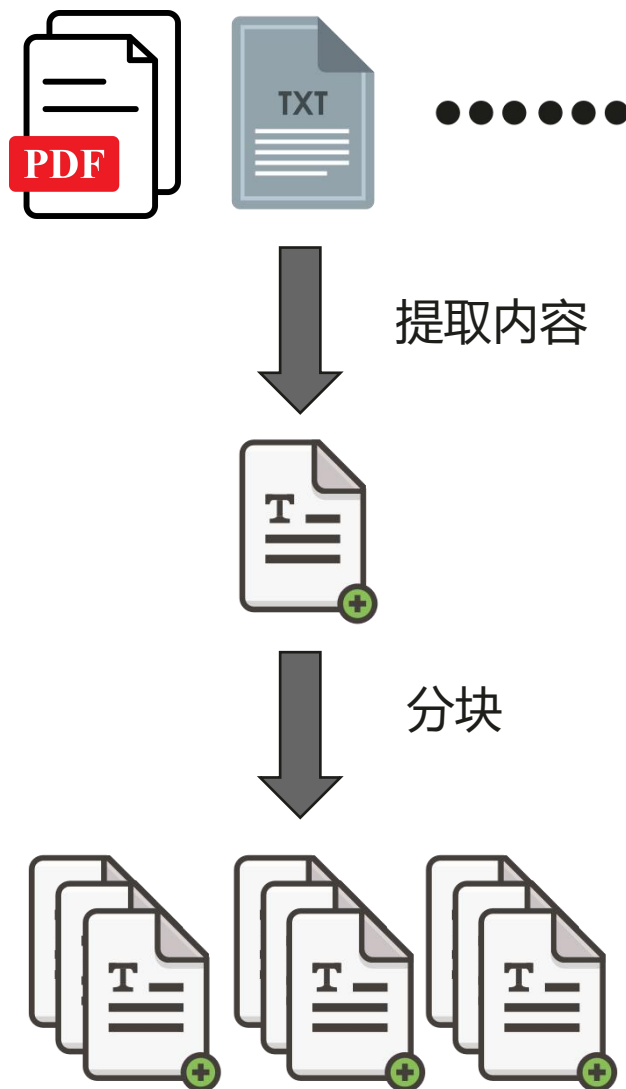
- **向量化**：用户提出问题时，将问题文本进行同样的向量化处理，得到问题的向量表示。
- **检索**：通过向量数据库进行相似度检索(eg.余弦相似度)，从知识库中找到与用户问题最相关的文档切片。
- **响应**：检索到相关文档切片，LLM会根据检索到的文本块作为额外的上下文，加上用户的问题生成答案。

实战篇：从零开始实现 MindRAG

1. 读取文件与处理

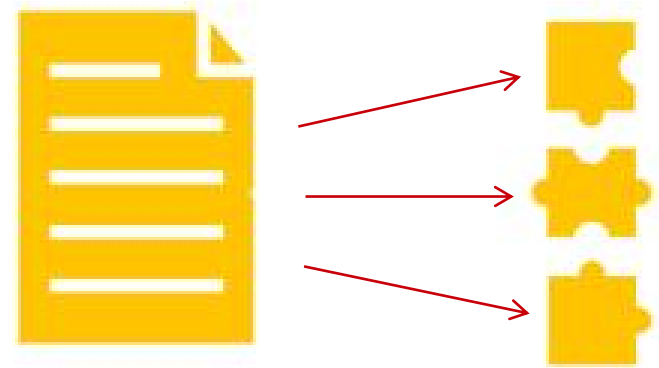
需求分析：

1. 读取文件：读取对应文件夹下所有文件。
2. 提取内容：判断文件类型，设计提取内容方式，实现多种格式统一化处理。
3. 分块：采用基于最大 token 长度和覆盖内容的逻辑分割长文本，确保段落间的语义连续性。



1. 读取文件与处理：分块策略

- 基于固定大小的分块 (Fixed-Size Chunking)
- 基于段落的分块 (Paragraph-Based Chunking)
- 基于句子的分块 (Sentence-Based Chunking)
- 基于语义的分块 (Semantic Chunking)
- 基于滑动窗口的分块 (Sliding Window Chunking)
- 基于主题的分块 (Topic-Based Chunking)
- 混合分块策略 (Hybrid Chunking Strategies)
-



1. 读取文件与处理：滑动窗口分块

滑动窗口分块是一种常见的文本分块技术，特别适用于需要在分块之间保留上下文信息的场景。它通过在分块之间引入部分重叠内容（即滑动窗口的重复区域），在一定程度上保持跨块的语义连贯性。通过这种方式，每个分块不仅包含当前段落的信息，还共享了一部分前后文内容。

窗口大小 (chunk size)：定义每个分块的最大长度，通常以字符、单词或令牌为单位。

滑动步长 (stride size)：定义分块的起始位置之间的距离。从而实现分块之间的重叠。

重叠内容 (overlap)：滑动窗口的重叠部分由 $|\text{chunk size} - \text{stride size}|$ 决定，用于保留上下文。

分块流程:

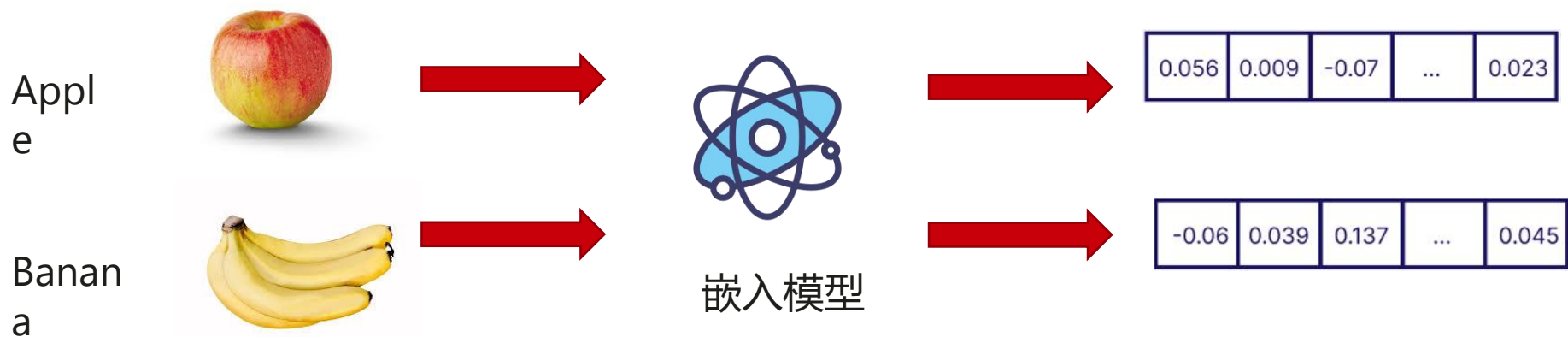
1. 确定每块的最大长度（窗口大小）。
2. 确定滑动的步长，计算分块之间的重叠区域。
3. 从文档开头开始，提取长度为窗口大小的内容作为第一个分块。
4. 从前一个分块的起始位置向前滑动步长，提取下一段内容。
5. 重复步骤4，直到处理完整个文档。

1. 读取文件与处理：混合分块策略

在我们的RAG框架中，我们结合了**固定大小的分块**和**滑动窗口分块**等策略，尽可能保证语义的连贯和一致性。具体而言，我们的策略如下：

- 对于一个文档，我们以**换行符**作为标志将一行作为一个段落。
- 如果某一行(段落)没有达到固定大小，我们不对其进行分块。
- 如果某一行(段落)大于固定大小，我们使用**滑动窗口分块**策略进行分块，保证语义的连贯性。
- 避免直接在单词中间切割。

2. Embedding设计： 嵌入模型



在RAG中，嵌入模型（Embedding Model）是用于将文本数据转化为高维向量的工具。这些向量是文本的稠密表示，能够捕捉文本的语义信息。嵌入模型的核心作用是使语义上相似的文本在向量空间中靠近，从而便于检索或匹配。

2. Embedding设计：检索

怎么去评估一个文本和另一段文本相关呢？



怎么去评估一个文本的Embedding和另一段文本的Embedding相关呢？



余弦相似度



余弦相似度通过计算两个向量之间的夹角的余弦值，来表示它们在向量空间中的相似性。余弦相似度的值范围在 $[-1, 1]$ 之间。



$$\cos \theta = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \cdot \|\mathbf{B}\|}$$



```
def cosine_similarity(cls, vector1: List[float], vector2: List[float]) -> float:
    """
    calculate cosine similarity between two vectors
    """
    dot_product = np.dot(vector1, vector2)
    magnitude = np.linalg.norm(vector1) * np.linalg.norm(vector2)
    if not magnitude:
        return 0
    return dot_product / magnitude
```

mindspore-lab/mindnlp: Easy-to-use and high-performance NLP and LLM framework based on MindSpore, compatible with models and datasets of ??????Huggingface.

2. Embedding设计: SentenceTransformer

MindNLP 是基于 MindSpore 开源的 NLP 库，支撑解决自然语言处理任务的平台，涵盖了 NLP 中很多常用的方法，可以帮助研究开发者更加便捷地构建和训练模型。

在MindNLP提供了 SentenceTransformers，这是一个用于句子、文本嵌入的组件，提供了简单易用的接口来生成高质量的文本嵌入。

SentenceTransformers的主要目标是使得生成文本嵌入变得更加简单和高效，特别适用于需要比较句子或文本相似度的任务，如语义搜索、聚类、文本匹配等。

```
from mindnlp.sentence import SentenceTransformer

sentences_1 = "今天天气真好，适合出去散步。"
sentences_2 = "天气不错，适合外出散步。"

model = SentenceTransformer('BAAI/bge-large-zh-v1.5')
embeddings_1 = model.encode(sentences_1, normalize_embeddings=True)
embeddings_2 = model.encode(sentences_2, normalize_embeddings=True)

similarity = embeddings_1 @ embeddings_2.T
```



3. 知识库设计： 常用向量数据库

在RAG架构设计中，知识库通常使用向量数据库搭建。

向量数据库是一种专门用于存储、索引和检索高维向量数据的数据库系统，它通过高效的相似性搜索功能，能够在大量向量数据中快速找到与查询向量最相似的向量，加速了检索的速度

FAISS :由 Facebook AI Research 开发。它支持多种向量索引结构，如倒排文件 (IVF)、乘积量化 (PQ) 等，能够在大型数据集上进行快速检索。



Milvus: 开源的向量数据库，专门用于处理大规模向量数据的存储和检索。它支持多种索引类型，如 IVF、HNSW、Annoy 等，并且提供了分布式部署的能力。

GaussDB for Vector: 华为云数据库 GaussDB 系列，专门针对高维向量数据的存储和检索进行了优化。它提供了高效的相似性搜索功能，支持多种索引结构，能够在大型数据集上实现快速的近似最近邻搜索。

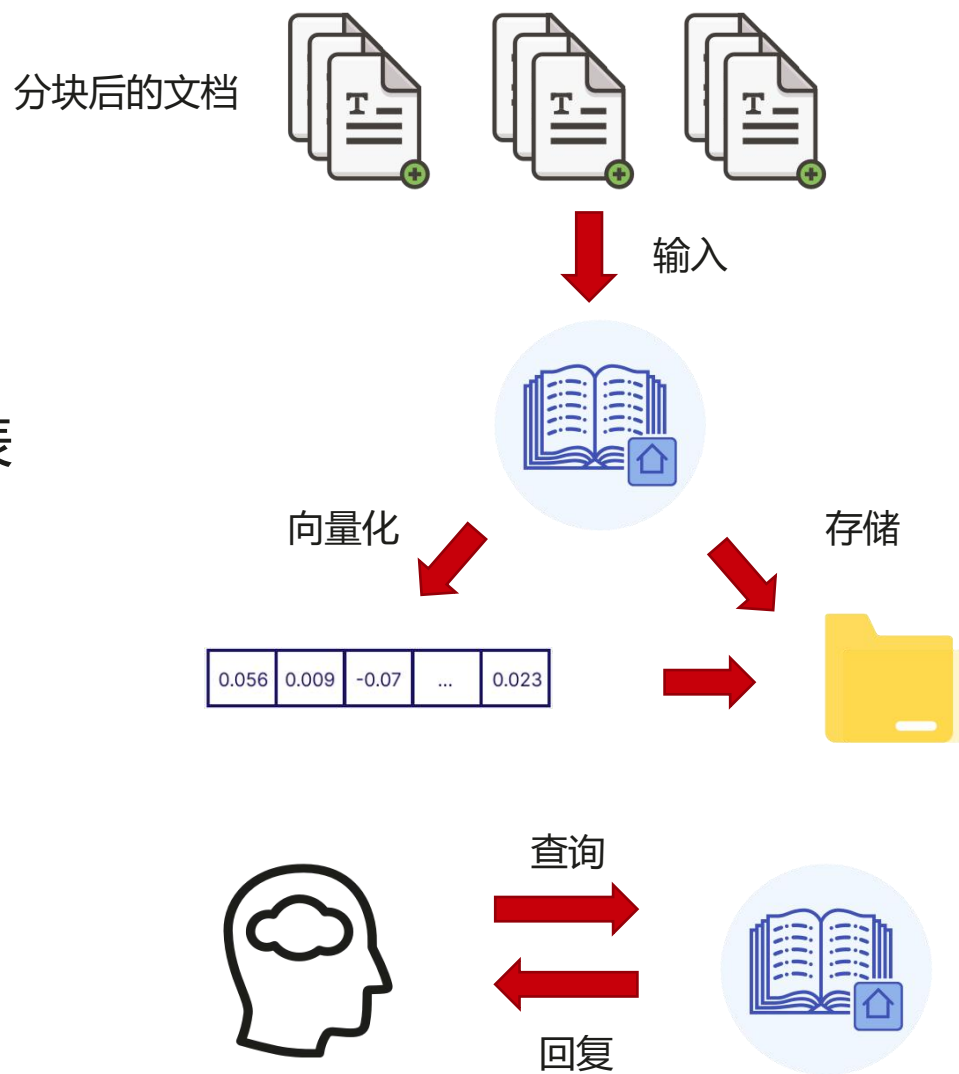
GaussDB



3. 知识库设计

需求分析：

1. **嵌入向量计算**：通过提供的嵌入模型，将文档转换为向量形式，以便对文档内容进行向量化表示。
2. **数据持久化**：可以将文本文档和嵌入向量存储到本地，便于重启系统时继续使用之前的数据。
3. **从文件中加载数据**：实现从本地加载持久化保存的文档和向量，恢复系统状态的功能。
4. **相似文档查询**：支持根据用户输入的查询文本找到与之最相似的文档。



4. 大语言模型：对话模型

在模型选择上，我们选择MiniCPM-2B大语言模型

MiniCPM 是面壁与清华大学自然语言处理实验室共同开源的系列端侧语言大模型，主体语言模型 MiniCPM-2B 仅有 24亿（2.4B）的非词嵌入参数量。

MiniCPM 在公开综合性评测集上，MiniCPM 与 Mistral-7B相近（中文、数学、代码能力更优），整体性能超越 Llama2-13B、MPT-30B、Falcon-40B 等模型。

我们使用MindNLP加载MiniCPM模型，MindNLP无缝兼容HuggingFace模型库，本地模型支持**250+ transformers模型**，覆盖90% huggingface模型。这提供了丰富的预训练模型和算法，可以快速构建和部署NLP应用。

易用性上，MindNLP拥有与Transformers库类似的接口，便于使用和学习。

MindNLP针对不同硬件平台进行了深度优化，支持CPU、GPU、Ascend等多种硬件加速，能够充分发挥硬件性能，实现高效的模型推理。

4. 大语言模型：提示设计

```
MindNLP_PROMPT_TEMPLATE="""先对上下文进行内容总结,再使用上下文来回答用户的问题。如果你不知道答案,请输出我不知道。总是使用中文回答。

问题: {question}
可参考的上下文:
...
{context}
...

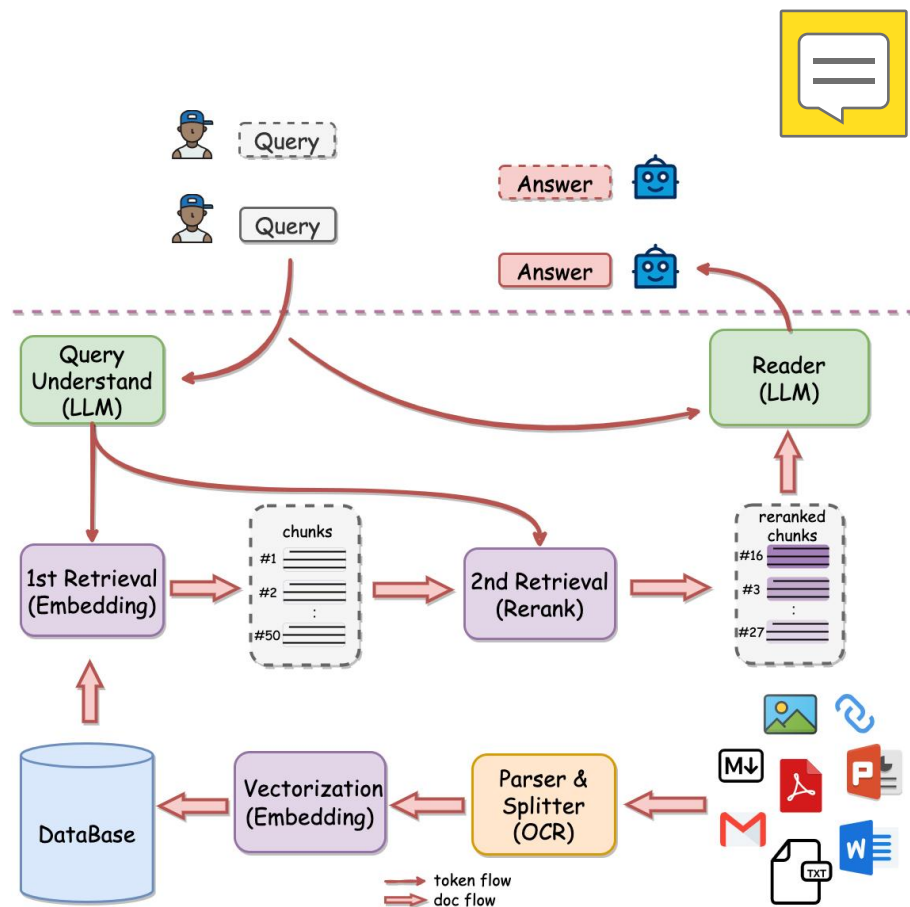
如果给定的上下文无法让你做出回答,请回答数据库中没有这个内容,你不知道。
有用的回答: """
```

在RAG的prompt设计中,关键是如何有效地引导模型进行信息检索和生成:

- **明确检索目标:** 通过明确指出需要检索的文档内容和用户问题,引导模型首先进行信息检索,然后再生成答案。
- **提供上下文信息:** 提供上下文信息可以帮助模型更好地理解检索到的文档内容,并生成更相关的文本。
- **减少幻觉:** 确保模型在无法回答时能够给出明确的反馈。



5. Rerank: 定义



在检索增强生成（RAG）系统中，rerank算法（重排序算法）是一种用于优化检索结果的技术。在检索阶段，系统从大规模的文档库中检索出与用户查询相关的文档或段落。然而，初始检索结果可能并不总是最优的，因为检索模型可能受到噪声、不相关文档或排序不准确的影响。

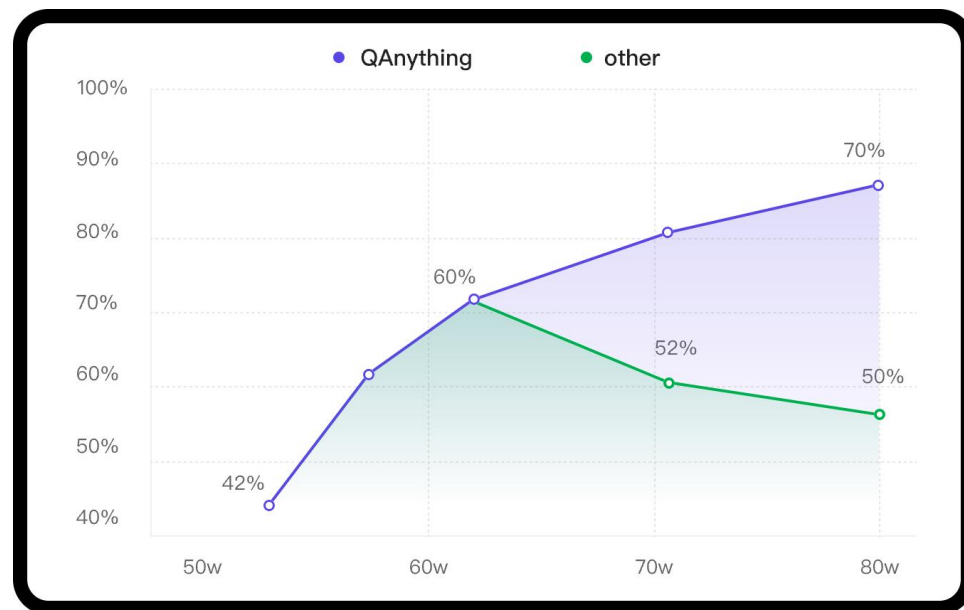
为了提高检索结果的质量，rerank算法被引入。rerank算法的作用是对初始检索结果进行重新排序，以确保最相关和最有用的文档或段落排在前面。

简单来说，在第一次检索完成后，使用一个更精细化的嵌入模型(Reranker)进行重新排序，选择最相关的文本。

5. Rerank: 为什么需要Rerank

知识库数据量大的场景下Rerank优势非常明显，如果只用一阶段embedding检索，随着数据量增大会出现检索退化的问题(Lost In Middle现象)。

这个问题是由于context window的限制，向量检索返回的候选chunk有限，与query相关的信息可能会被排除在候选集之外。在两阶段检索中，第一阶段适当增大候选集，使得相关chunk都被检索到；第二阶段利用reranker，将query和候选chunk一起输入transformer模型计算相似得分，使得相关chunk排序靠前，然后将重排序后更小的候选集输入给LLM。从实验结果来看，reranker可以有效地提升RAG的召回和回答准确性。



5. Rerank



```
def rerank(self, text: str, content: List[str], k: int) -> List[str]:
    query_embedding = self._model.encode(text, normalize_embeddings=True)
    sentences_embedding = self._model.encode(sentences=content, normalize_embeddings=True)
    similarity = query_embedding @ sentences_embedding.T
    # 获取按相似度排序后的索引
    ranked_indices = np.argsort(similarity)[::-1] # 按相似度降序排序
    # 选择前 k 个最相关的候选内容
    top_k_sentences = [content[i] for i in ranked_indices[:k]]
    return top_k_sentences
```

Rerank和第一次检索的过程基本一致：

1. 获取候选集(第一次检索出)的句子嵌入(文本语义特征)
2. 计算Query和候选集之间的相似度
3. 选择前K个候选内容返回

6. Demo



Gradio 是一个开源的 Python 库，用于快速创建和分享机器学习模型的交互式界面。它允许开发者通过简单的代码将模型、函数或数据处理流程转化为易于使用的 Web 应用，支持输入输出组件（如文本框、图像、滑块等），并自动生成用户界面。Gradio 特别适合用于演示、测试和部署机器学习模型，无需复杂的前端开发，即可让用户通过浏览器与模型交互。它还支持一键分享，方便团队协作或对外展示。

作业

在基础的RAG demo上，加入Rerank机制