LLM RAG

https://arxiv.org/abs/2312.10997

《Retrieval-Augmented Generation for Large Language Models: A Survey》这篇论文介绍了 RAG 技术的发展。

本文尝试对其关键要点进行一些简单的介绍。

Why Need RAG, LLM 面临的问题

在介绍 RAG 是什么以前，先介绍了 LLM 目前所面临的问题

训练成本

神经网络类型的 AI 有一大特点，就是预训练的成本远远高于推理。

OpenAI 在 Dev Day 2023 宣称 GPT 的预训练成本在 2～3 百万美元。

数据集落后

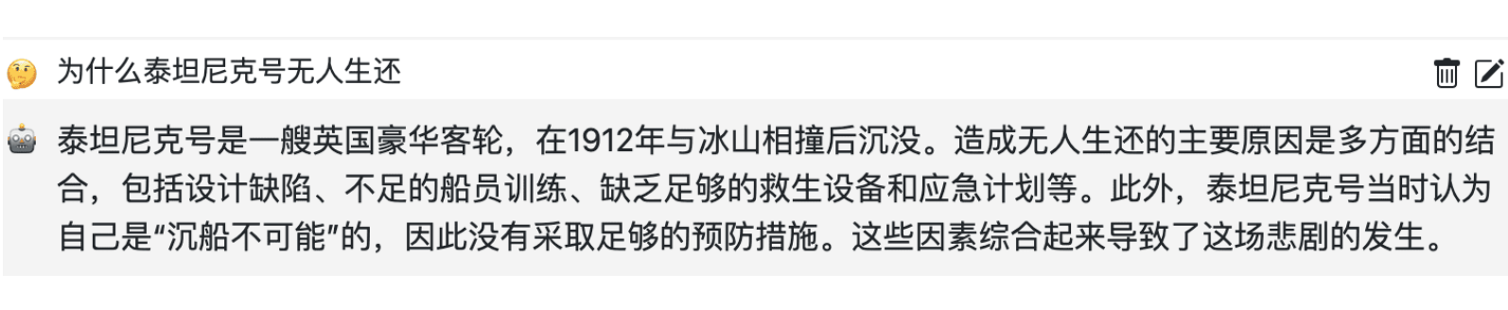
高昂的预训练成本带来一个最直接的问题就是：模型更新缓慢。 GPT-3.5 的数据集时间为 September 2021。

而且数据集的更新成本不仅仅是训练，还有数据集的收集和清洗，这都进一步降低了模型更新的频率。

hallucination 幻觉

有一种观点认为 LLM 是一种高效的有损信息压缩算法，它的信息解压缩过程依赖于用户 prompt。

这使得 LLM 的答复质量非常容易受到用户 prompt 的干扰。



为什么泰坦尼克号无人生还

泰坦尼克号是一艘英国豪华客轮，在1912年与冰山相撞后沉没。造成无人生还的主要原因是多方面的结合，包括设计缺陷、不足的船员训练、缺乏足够的救生设备和应急计划等。此外，泰坦尼克号当时认为自己是“沉船不可能”的，因此没有采取足够的预防措施。这些因素综合起来导致了这场悲剧的发生。

Transparency 透明性

LLM 给出的回答完全是黑盒，根本不知道来自哪，自然也就难以查证。

What is RAG

Parameter

我们将 LLM 的信息分为两个渠道：

1. Parameteric knowledge：预训练 LLM 时使用的信息
2. Non-parametric knowledge：LLM 推理时 context 内的信息

RAG

根据 LLM 对 parameter 的依赖程度，可以再分为三类：

1. fully parameterized model：只依赖预训练数据
2. RAG：hybrid
3. RCG：完全依赖推理时的外部信息

所以可以对 RAG 进行定义：Retrieval-Augmented Generation，基于信息抓取的生成。

或者更通俗的理解为：在直接将数据交给模型以前，先进行一轮信息检索，完善输入信息。

RAG Vs. Fine-Tunning

RAG 和 fine-tuning 都是可以提高 LLM 模型性能的方法，两者的应用场景存在一些差别：

1. RAG 适合少量垂直领域的精确信息，不适合开放式的大量数据
2. fine-tuning 适合大量数据

需要注意的是，fine-tuning 和 pre-traning 的区别在于：

fine-tuning 不适合让 LLM 学习新知识，而是适合让 LLM 强化某个已知知识。

可以认为 fine-tuning 是复习，RAG 是考试时的小抄。

What is fine-tuning

其实就是组织大量结构化的问答数据喂给 LLM，增强其对特定语料的回答能力。

```
{
    "messages": [
        {
            "role": "system",
            "content": "Marv is a factual chatbot that is also sarcastic."
        },
        {
            "role": "user",
            "content": "What's the capital of France?"
        },
        {
            "role": "assistant",
            "content": "Paris, as if everyone doesn't know that already."
        }
    ]
}
```

fine-tuning 和 RAG 并不矛盾，两者契合可以发挥出更大的作用

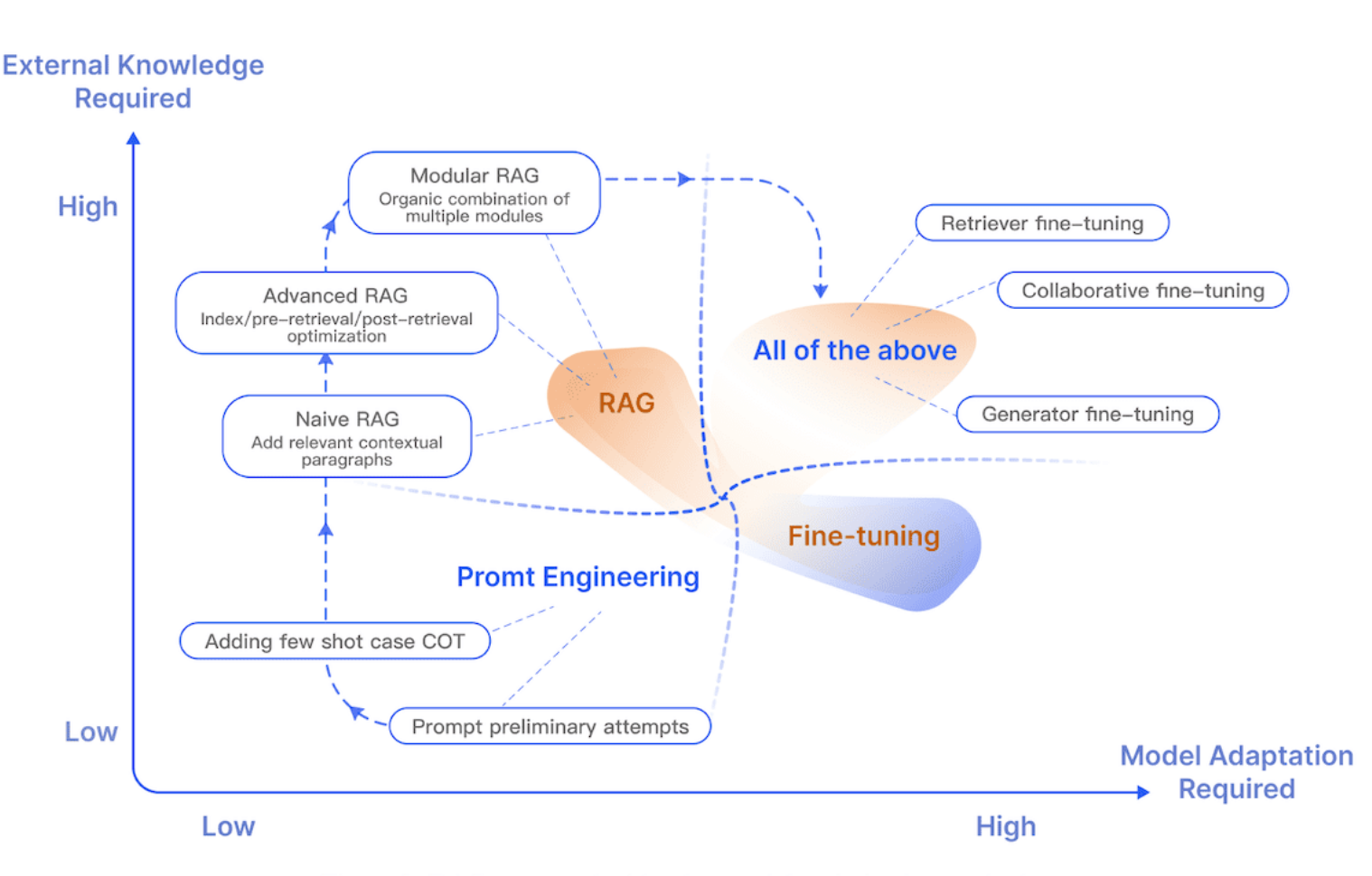
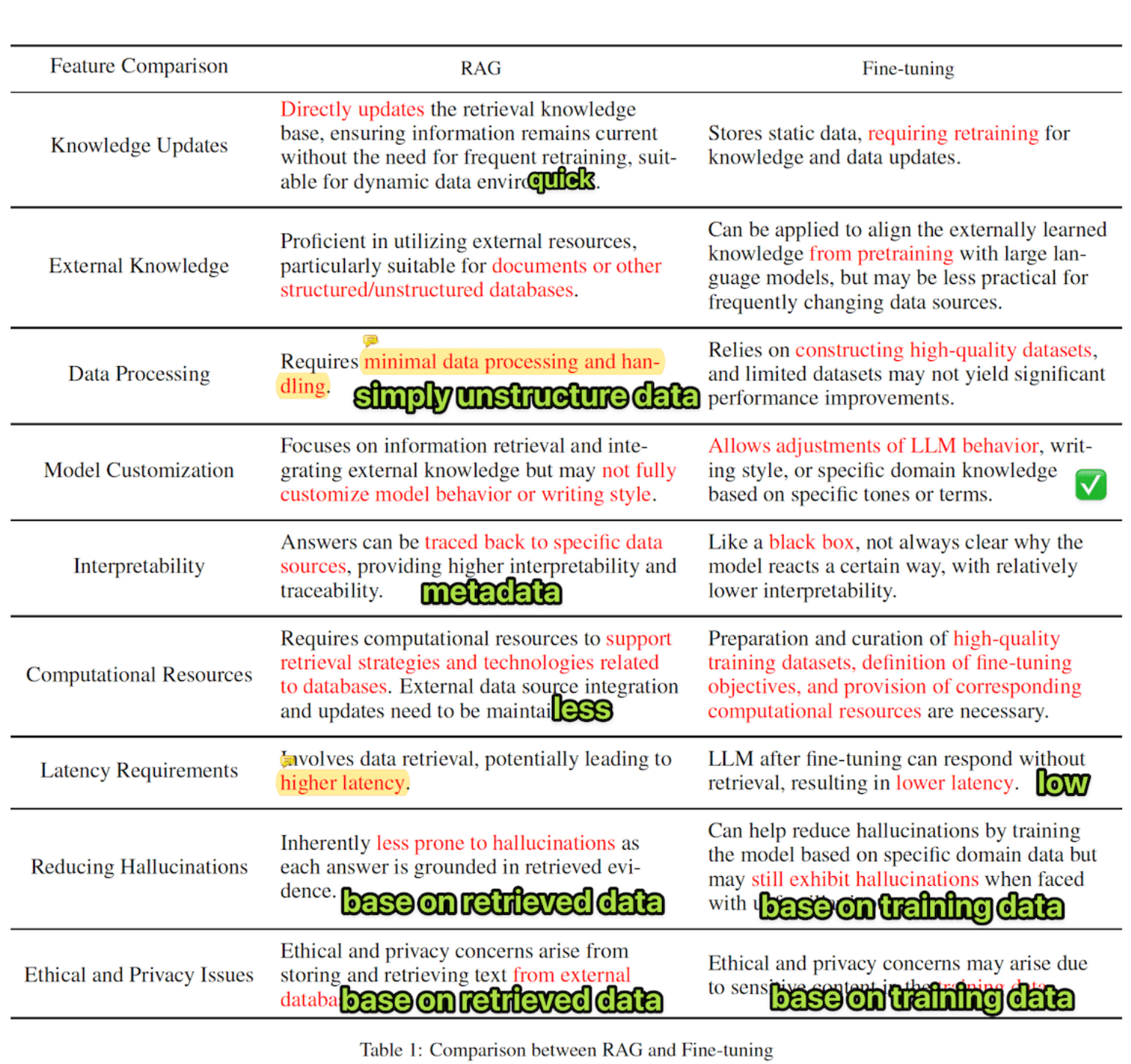


Figure 2: RAG compared with other model optimization methods



Feature Comparison	RAG	Fine-tuning
Knowledge Updates	Directly updates the retrieval knowledge base, ensuring information remains current without the need for frequent retraining, suitable for dynamic data envir quick.	Stores static data, requiring retraining for knowledge and data updates.
External Knowledge	Proficient in utilizing external resources, particularly suitable for documents or other structured/unstructured databases.	Can be applied to align the externally learned knowledge from pretraining with large language models, but may be less practical for frequently changing data sources.
Data Processing	Requires minimal data processing and handling. simply unstructure data	Relies on constructing high-quality datasets, and limited datasets may not yield significant performance improvements.
Model Customization	Focuses on information retrieval and integrating external knowledge but may not fully customize model behavior or writing style.	Allows adjustments of LLM behavior, writing style, or specific domain knowledge based on specific tones or terms. ✅
Interpretability	Answers can be traced back to specific data sources, providing higher interpretability and traceability. metadata	Like a black box, not always clear why the model reacts a certain way, with relatively lower interpretability.
Computational Resources	Requires computational resources to support retrieval strategies and technologies related to databases. External data source integration and updates need to be maintai less	Preparation and curation of high-quality training datasets, definition of fine-tuning objectives, and provision of corresponding computational resources are necessary.
Latency Requirements	Involves data retrieval, potentially leading to higher latency.	LLM after fine-tuning can respond without retrieval, resulting in lower latency. low
Reducing Hallucinations	Inherently less prone to hallucinations as each answer is grounded in retrieved evidence. base on retrieved data	Can help reduce hallucinations by training the model based on specific domain data but may still exhibit hallucinations when faced with base on training data
Ethical and Privacy Issues	Ethical and privacy concerns arise from storing and retrieving text from external databa base on retrieved data	Ethical and privacy concerns may arise due to sens base on training data

Table 1: Comparison between RAG and Fine-tuning

RAG's Evolution

介绍 RAG 技术和工作流的进化

Naive RAG

最简单的 RAG，就是顾名思义的执行三个步骤：

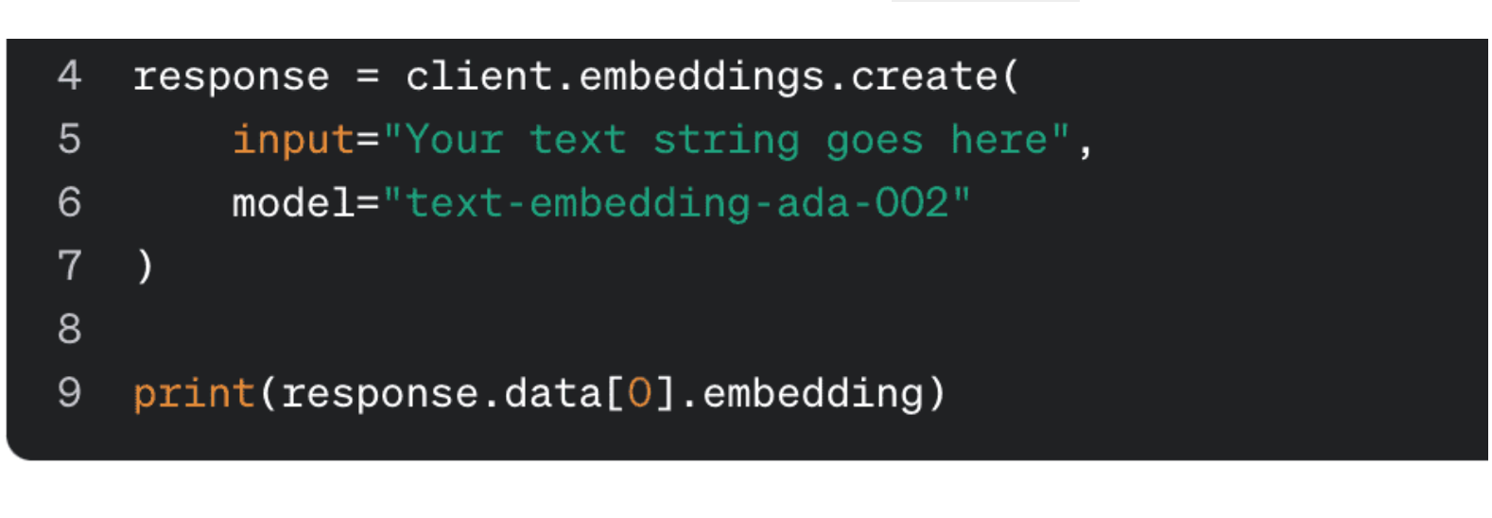
1. Retrieval: 根据 prompt 抓取外部数据
2. Augmented: 使用外部数据增强 prompt
3. Generation: 把增强后的 prompt 交给 LLM，生成 predict/answer

embeddings

Naive 处理资料集的方式也是单一的：

1. 首先，全部转换为文字（text）
2. 对文字进行切块（chunk）
3. 把每一个 chunk 交给 embeddings-model，计算词向量（word vector）
4. 将词向量和 chunk 存储向量数据库

embeddings 就是基于 LLM 将一个语句转化为一个高维向量，切分资料的 `chunk_size` 是一个关键参数



```
response = client.embeddings.create(
    input="Your text string goes here",
    model="text-embedding-ada-002"
)

print(response.data[0].embedding)
```

Example response:

```
{
  "data": [
    {
      "embedding": [
        -0.006929283495992422,
        -0.005336422007530928,
        ...
        -4.547132266452536e-05,
        -0.024047505110502243
      ],
      "index": 0,
      "object": "embedding"
    }
  ],
```

Retrieval

执行类似步骤：

1. 将 prompt 交给 embeddings-model，计算词向量
2. 在向量数据库中执行 KNN 查询，获取 K 个最近似结果

Augmented

将获取到的资料按照某个模板，和 prompt 进行拼接，得到最终 prompt

Generation

将最终 prompt 交给 LLM，得到回答。

一个简单的结合向量数据库实现 prompt 增强的例子：

```
embeddings = OpenAIEmbeddings()

# Vector Store
db = Chroma.from_documents(documents=texts, embedding=embeddings)

# Create Prompt
template = """Use the following pieces of context to answer the question at the end.
If you don't know the answer, just say that you don't know.
Don't try to make up an answer.
{context}

Question: {question}
Answer:
```

向量化的问题

可以看出数据的检索完全依赖于向量化所产生的词向量。

> when an embedding model calculates the vector representation of a sentence, it does so based on the similarity of the sentence to the pre-trained data

而 embedding model 在计算一个语句的向量时，需要基于 pre-training 的数据来理解输入的语句。如果输入的语句和预训练数据集的差异特别大，会导致结果出现很大偏差。

基于词向量相似性搜索的方案，precision 和 recall 都很低。就是查询的数据不一定有用，有用的数据不一定被查询到。

Ps. precision 度量结果数据中的阳性率。recall 度量所有阳性被找到的概率。都是越高越好。

Advanced RAG

Advanced RAG 在 Naive 的基础上增加了 pre-retrieval 和 post-retrieval 两个方法。

工作流变为：

1. pre-retrieval process
2. embeddings
3. post-retrieval process

Pre-retrieval Process

在 embeddings 以前，对数据进行清洗和规整，可以分为 5 个步骤：

1. Enhancing Data Granularity: 对数据内容进行修订和简化，确保数据源的正确性和可读性
2. Optimizing Index Structures: 优化数据索引，引入图数据库等关联结构
3. Adding Medadata Information: 为切块后的数据增加 metadata，标记数据来源
4. Alignment Optimization: 可以为每一个 chunk 生成一个假设性提问，然后将这个问题本身也嵌合到 chunk 中，这样可以提高检索的关联度。
5. Mixed Retrieval: 混合使用多种检索技术，而不仅仅是词向量搜索。

Embedding

对 embeddings 过程中所使用的 embedding-model 也进行改进

1. Fine-tunning Embedding: 可以将领域知识预先通过 fine-tuning 内嵌到模型中
2. Dynamic Embedding: 在 embeddings 时，不要仅针对关键词（static），而是要联合上下文一起（dynamic）。

Post-Retrieval Process

在完成资料查询，提交给 LLM 前，继续对收集到的资料进行优化

1. ReRank: 根据关联度进行打分和重排序
2. Prompt Compression: 无关输入对 LLM 的性能有负面影响。压缩不相关信息，强调关键信息，减少总长度

RAG Pipeline Optimization

一些通用的 RAG 检索资料优化办法

1. Hybrid Search: 前面提到过的混合搜索
2. Recursive Retrieval And Query Engine: 多阶段检索，先检索一批小 chunk，再根据小 chunk 去检索大 chunk
3. StepBack-prompt: 一种 prompt-engineering，可以显著提高推理密集型任务的性能。让 LLM 更关注抽象概念
4. Subqueries: 根据语意拆分为多个小查询
5. HyDE: 先让 LLM 回答一次，然后根据 LLM 的回答再去搜索相关资料。但如果 LLM 对相关话题不熟悉，反而会加重幻觉。

Modular RAG

这是作者提到的 RAG 的最终进化形态。不过其实所使用的技术都是前面 naive 和 advanced 里提到过的。

最重要的改变更多是架构设计上的，将单一的命令式流水线（pipeline），变成了相应式的动态调度。

所有前面提到过的功能都被封装为了功能模块，根据任务类型进行动态组合和调度。

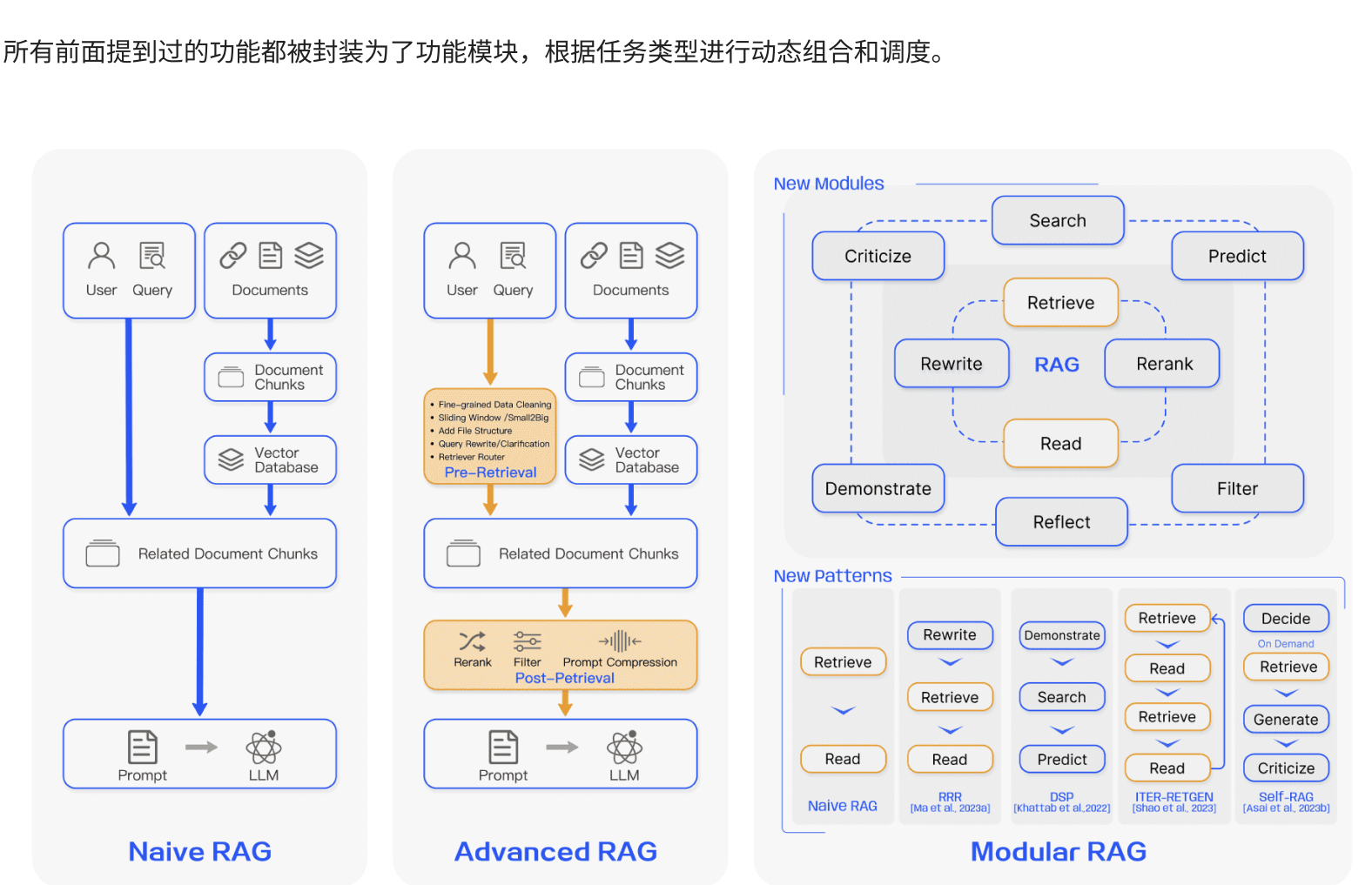


Figure 3: Comparison between the three paradigms of RAG

Retriever

在 Modular RAG 中，Retriever 负责对外部数据源进行预处理和查询。

增强 Embeddings 的语意准确度

Chunk

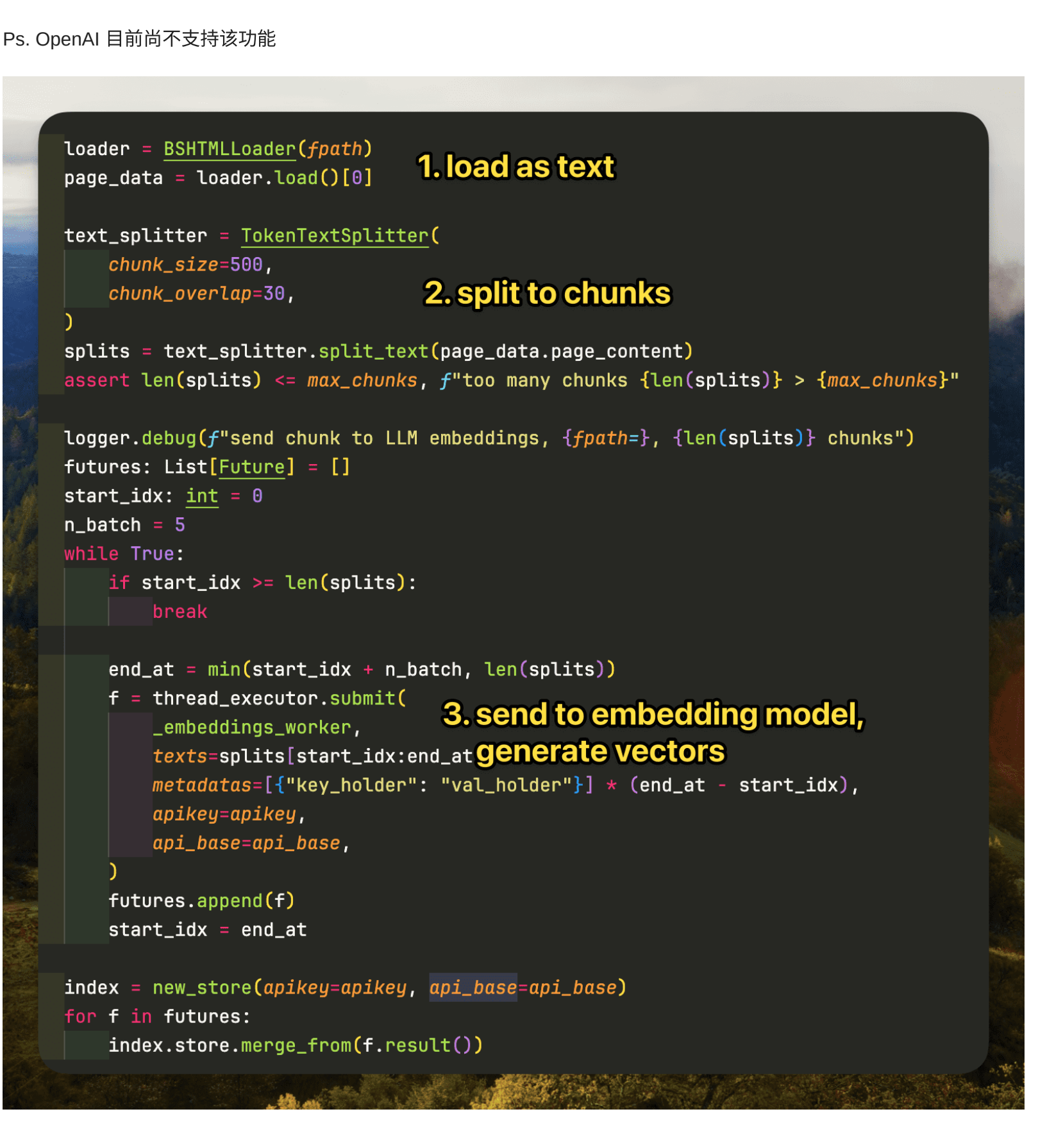
RAG 的最终生成物不能超出 LLM 的 context window，所以数据切片是必要的，而切片的 chunk size 的选择是优化的第一步。

OpenAI 的 text-embedding-ada-002 的最优 chunk size 为 256～512 tokens。

Fine-tuning Embedding Models

embedding 是 RAG 的核心，为了让 embedding model 能够更好地理解垂直领域信息，可以对 embedding model 进行 fine-tuning。

Ps. OpenAI 目前尚不支持该功能



```
loader = BSHTMLLoader(fpath)
page_data = loader.load()[0]

text_splitter = TokenTextSplitter(
    chunk_size=500,
    chunk_overlap=30,
)
splits = text_splitter.split_text(page_data.page_content)
assert len(splits) <= max_chunks, f"too many chunks {len(splits)} > {max_chunks}"

logger.debug(f"send chunk to LLM embeddings, {fpath=}, {len(splits)} chunks")
futures: List[Future] = []
start_idx: int = 0
n_batch = 5
while True:
    if start_idx >= len(splits):
        break

    end_at = min(start_idx + n_batch, len(splits))
    f = thread_executor.submit(
        _embeddings_worker,
        texts=splits[start_idx:end_at],
        metadatas=[{"key_holder": "val_holder"}] * (end_at - start_idx),
        apikey=apikey,
        api_base=api_base,
    )
    futures.append(f)
    start_idx = end_at

index = new_store(apikey=apikey, api_base=api_base)
for f in futures:
    index.store.merge_from(f.result())
```

将用户请求和数据集进行对齐

Query Rewrite

最简单直观的方法就是让 LLM 重写用户查询，比如将其拆分为多个子查询。

Query Embedding Transformation

query rewrite 是粗粒度的，query 的 embedding 应该是细粒度的。

query 最终也需要被 embedding 然后再去搜索外部资料，处理 query 的 embedding model 也可以被 fine-tuning，以使其可以更好的匹配特定任务，尤其是使其可以更好的关联到结构化的数据

Aligning Retriever's Output and LLM's Preference

单纯的计算 retriever 的 hit rate（正确性）是不够的，因为可能查找的资料并不是 LLM 所需要的。 相比单纯的信息正确，LLM 更偏好于可读性更好的资料。

所以 retriever 还需要对齐（alignment），才能真正提高 RAG 的性能。

Generator

Generator 负责将 retriever 抓取的资料转化为更好的格式，喂给 LLM

Post-Retrieval Processing

retriever 抓取的资料可能过长或存在冗余。在 retriever 后，提交给 LLM 前，增加一步做数据清洗的 post-retrieval 步骤。

post-retrieval 的核心目标：信息压缩和结果重排（rerank）。

信息压缩的必要性：降低噪音，减少长度，增强 LLM 生成效率。（冗余信息会极大的干扰 LLM 的生成质量）

Rerank

RAG 里的 context 不是越多越好，实际上添加的上下文越多，LLM 的旧性能指标可能会降低。

`Catastrophic Forgetting`：LLM 学习新知识后会遗忘旧知识，导致回归性能降级。

rerank 可以改善 CF，通过将最相关的信息放在最前面，然后限制总的信息量。 我理解是，提供的新信息越少，对旧指标的干扰也就越少。

Augmentation in RAG

RAG 对 LLM 的提升是全面的，可以从三个维度来理解 RAG：

1. Stages: RAG 可以应用于 LLM 生命周期的全部三个阶段：pre-training、fine-tuning、inference
2. Data: 可以对数据进行增强
3. Process: RAG 的具体实操方法

RAG 概念概览

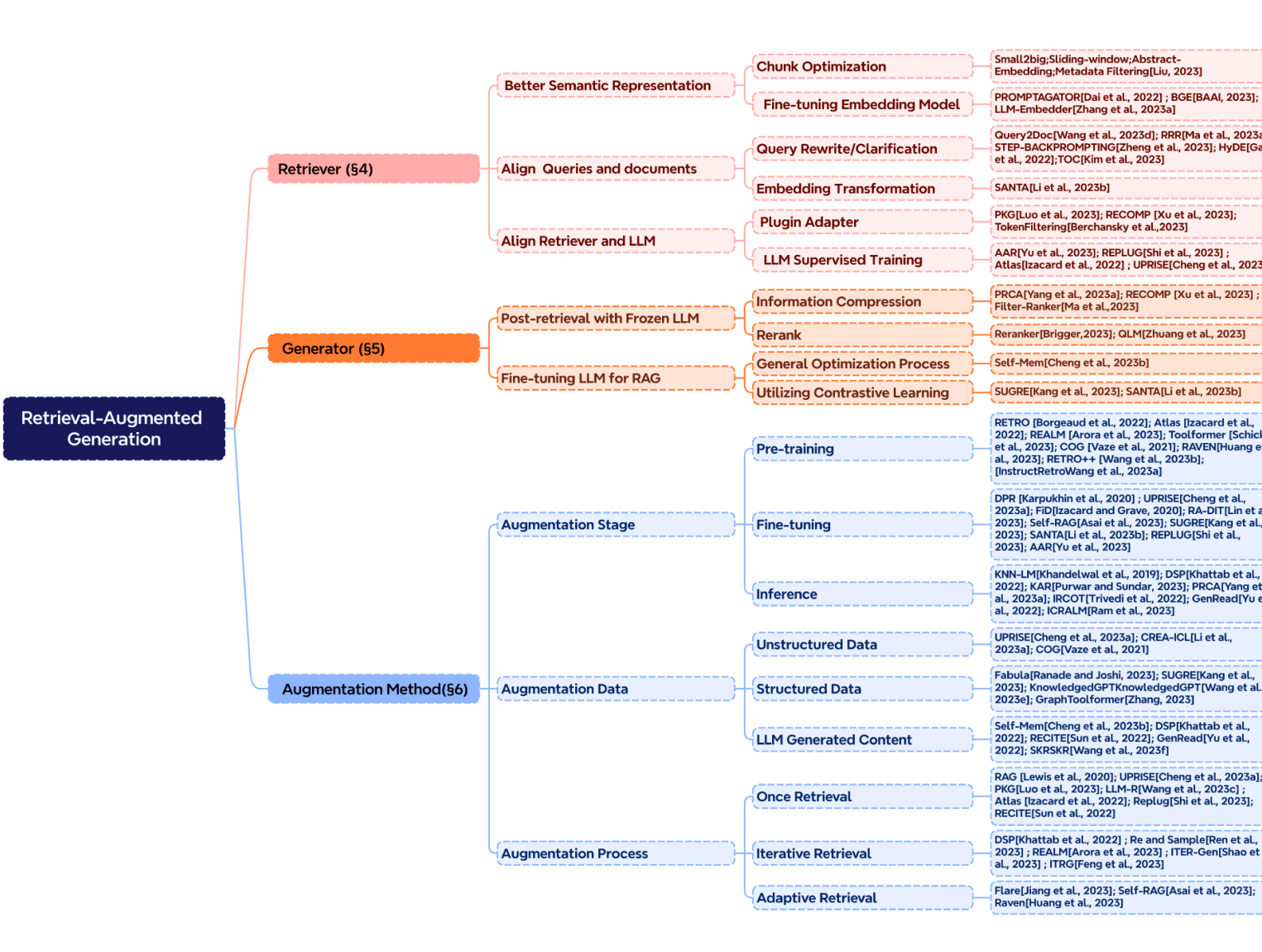
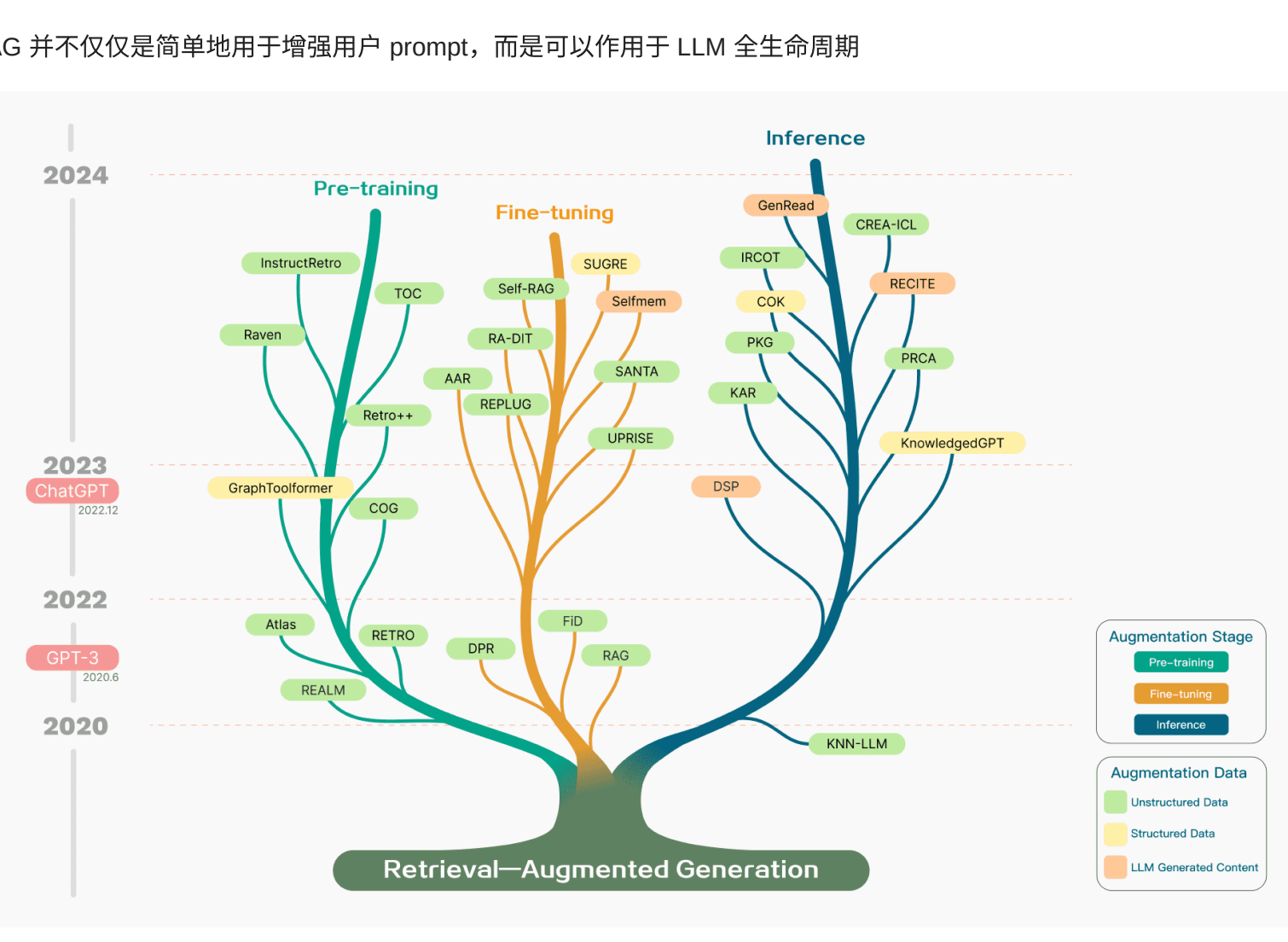


Figure 4: Taxonomy of RAG's Core Components

Stages

RAG 并不仅仅是简单地用于增强用户 prompt，而是可以作用于 LLM 全生命周期



Pre-training Stage

在 Pre-training 阶段就引入 RAG，对预训练数据集进行增强。

优点：基于 RAG 的数据增强，比从头去重新准备预训练数据集要更简单。

缺点：和预训练一样的缺点，更新缓慢，更新成本高。

Fine-tunning State

对于垂直领域，针对 LLM 和 retriever 都进行 fine-tuning，可以提高性能。

fine-tuning 的缺点：需要结构化的训练数据，需要的计算资源远大于推理。

Inference Stage

推理阶段 RAG 是目前最流行的方式

优点：轻量、便宜，实时

缺点：需要占据 LLM 宝贵的 context 空间，需要针对底层 LLM 进行定制化优化。

对数据源进行增强

基于知识图谱的结构化数据，可以在数据检索时提供更为关联的信息。

LLM Generated Content RAG

外部输入的新知识有时候会对 LLM 的性能带来负影响，所以有一条 RAG 的研究道路是探索深入挖掘 LLM 的内生知识。

SKR 让 LLM 区分已知和未知信息，只要求 retriever 获取未知信息，使用内置信息回答已知问题。

Augmentation Process

本节讨论 RAG 操作流程的优化。

单步 RAG 可能会因为信息冗余使得 LLM 抓不住重点（lost in the middle）。 此外，也让 LLM 无法进行深入的多步推理。

可以设计一个逻辑 loop: retriever - generator。通过多轮循环，以取得更好的效果。（Recursive retrieval and multi-hop retrieval）

Iterative Retrieval

利用外部数据增强推理 & 利用推理增强数据抓取，不断循环迭代

Adaptive Retrieval

实际上就是 tools/agents/function call 的功能，让 LLM 可以自行根据 context 调用 retriever 抓取外部数据源。

RAG Evaluation

Evaluation Methods

有两种评估方式，其实和我们惯常的测试方法论也是一样的

1. Independent Evaluation: 对各个流程/模块进行分别测试，可以理解为单元测试
2. End-to-End Evaluation: 顾名思义，模仿用户行为，直接就最终接口进行测试

E2E 测试又可分为 unlabeled 和 labeled。我的理解是，labeled 应该是偏向于人工校验，需要提供标准答案，然后计算 EM 等指标。unlabeled 偏向于自动测试，可以使用一些标准方法计算得分。

Retrieval Metrics

- `Hit Rate (HR)`：检索到相关资料的概率
- `Recall`：所有应该被检索的文档里，被正确检索出来的比例
- `Precesion`：检索出来的文档里，相关文档的比率
- `Mean Reciprocal Rank (MRR)`：rerank 的指标，度量 retriever 返回的最优信息是否出现在了最前列
- `Mean Average Precision (mAP)`：也称为 `mAP@K`，度量第 K 个正确答案的位置。可以理解为 MRR 的复数版。
- `Normalized Discounted Cumulative Gain (NDCG)`：度量整体的 rerank 质量
- `Exact Match (EM)`：查询到的资料里，包含正确答案的概率
- `F1 Score`：就是 recall 和 precision 的调和平均数：
- `Semantic Answer Similarity (SAS)`：比较正确答案和 LLM 回复（predict）间的语意相似度。

https://laisky.notion.site/Metrics-for-Information-Retrieval-and-Question-Answering-Pipelines-d5d4e3beb820419ca494596e319befcf?pvs=4

Key Metrics and Abilities

Key Metrics

1. Faithfulness: retriever 取回的资料不应该违背原始问题
2. Answer Relevance: RAG 生成 answer 应该忠于原始 query
3. Context Relevance: RAG 生成的 context 和原始 query 一致

Key abilities

能够支撑 RAG 的底层 LLM 所应该具备的四个基础能力：

1. Noise Robustness: 能够分辨噪音
2. Negative Rejection: 信息不足时应该拒绝回答
3. Information Integration: 能够整合杂乱的信息
4. Counterfactual Robustness:能够分辨 retrieval 提供的信息中存在的事实错误

明显在做梦😂，现实是通用大模型的 hallucination 非常显著

Future Prospects

RAG 未来的三个发展方向：

1. vertical optimization
2. horizontal expansion
3. ecosystem of RAG

Vertical Optimization

垂直优化，和水平扩容相对，一般指增强自身的能力，而不是依靠外部扩展，此处指增强 LLM 自身的能力。

包括更大的 context，更稳定的回答能力。目前的 LLM 极易受到参考信息中的错误和无关信息的干扰。

RAG 和 Fine-tuning 更深入地互相协调也是一个重要的优化方向。

最后就是 RAG 的工程实践，包括性能提升和安全性。

Horizontail Expansion

多模态！

Thanks