GraphRAG

RAG为llm提供了从某些数据源检索到的信息，作为其生成答案的依据。也就是将根据上下文相关信息进行检索，基于检索到的知识指导llm进行生成。

大模型面临的问题：

1. 幻觉：当没有答案时一本正经的胡说八道
2. 新鲜度：模型知识只截至到一个时间点，对之后的知识不了解
3. 隐私性：隐私知识的获取与训练成本高。

用知识图谱增强RAG能解决这些问题：

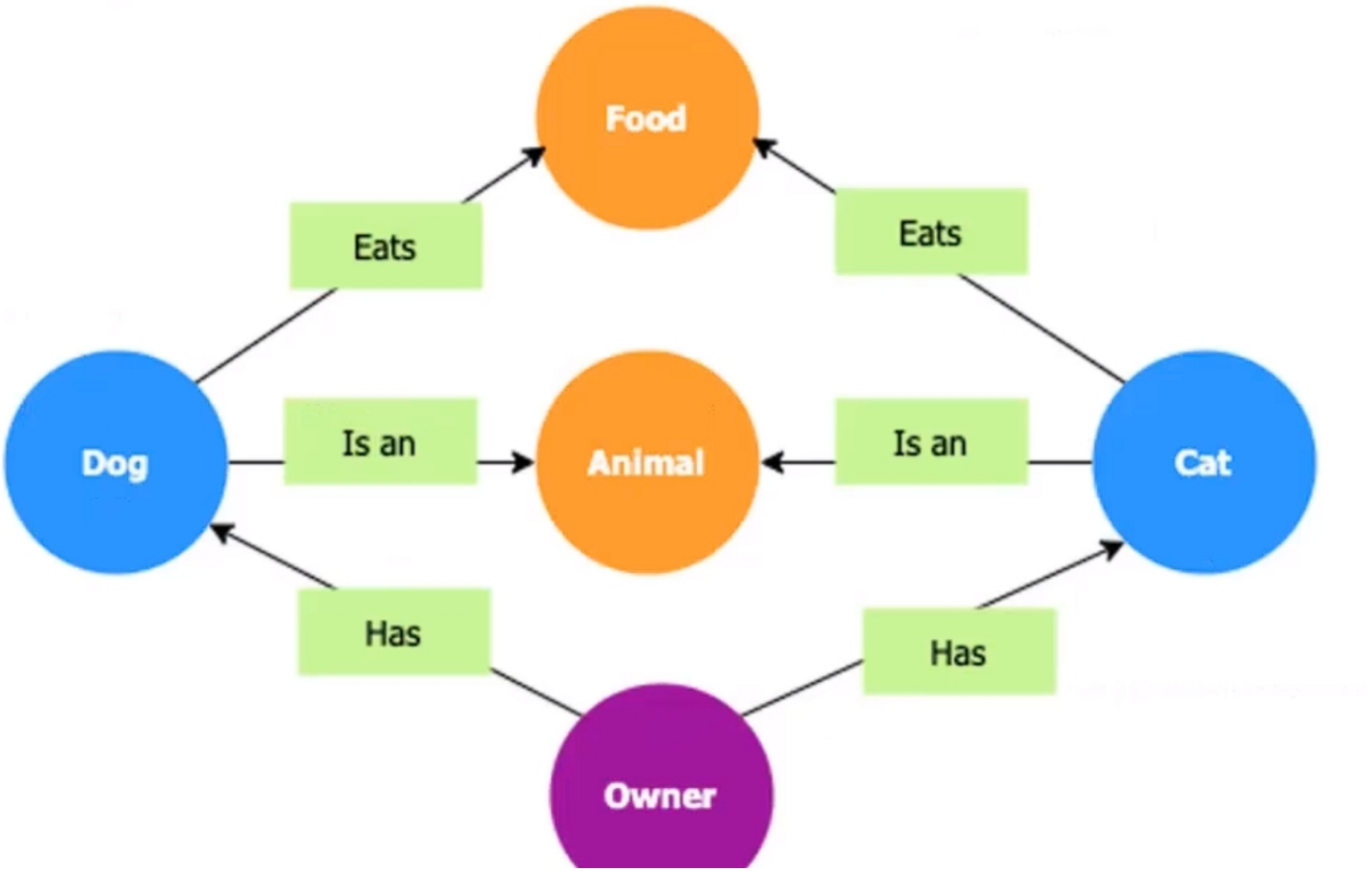
1. 减少生成内容幻觉，提供更多的可解释性。
2. RAG模型的非参数化记忆可以轻松更新，以反映下是世界的知识变化，无需对模型重新训练。
3. 通过参数化和非参数化的存储机制，增强nlp任务对知识的访问和操作能力。

知识图谱

定义：真实世界实体即其之间关系的结构化表示。

按照用途，知识图谱可以大致分为：百科全书式，多模态，常识性，特定领域

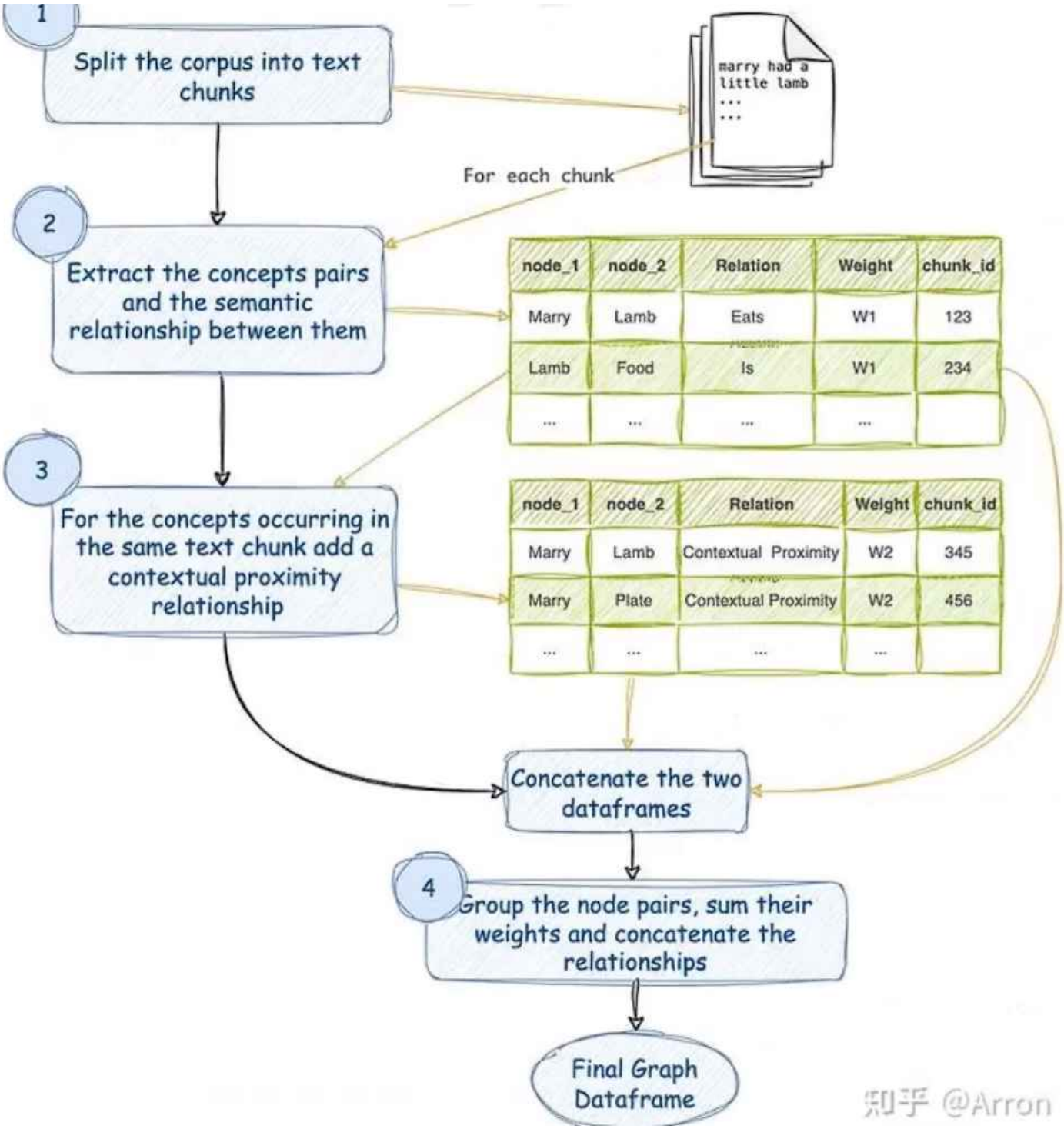
一般存储在图形数据库中，可以可视化的展示图形结构。点表示实体，边表示关系，例如：



图数据库可以分为：single-graph单边图和multi-graph多边图。例如A和B之间有多笔交易，如果采用单边图，但A和B之间就需要额外的点便是一笔交易，导致跳数计算时A和B的关系变远了；而多边图则不需要。

知识图谱的构建流程：

1. 将语料库拆分成文本块
2. 提取概念和语义联系。
3. 添加上下文邻接关系
4. 统一表示


1
Split the corpus into text chunks
marry had a little lamb
...
...
For each chunk
2
Extract the concepts pairs and the semantic relationship between them
node_1 node_2 Relation Weight chunk_id
Marry Lamb Eats W1 123
Lamb Food Is W1 234
... ...
3
For the concepts occurring in the same text chunk add a contextual proximity relationship
node_1 node_2 Relation Weight chunk_id
Marry Lamb Contextual Proximity W2 345
Marry Plate Contextual Proximity W2 456
... ...
Concatenate the two dataframes
4
Group the node pairs, sum their weights and concatenate the relationships
Final Graph Dataframe
知乎 @Arron

RAG面临的问题

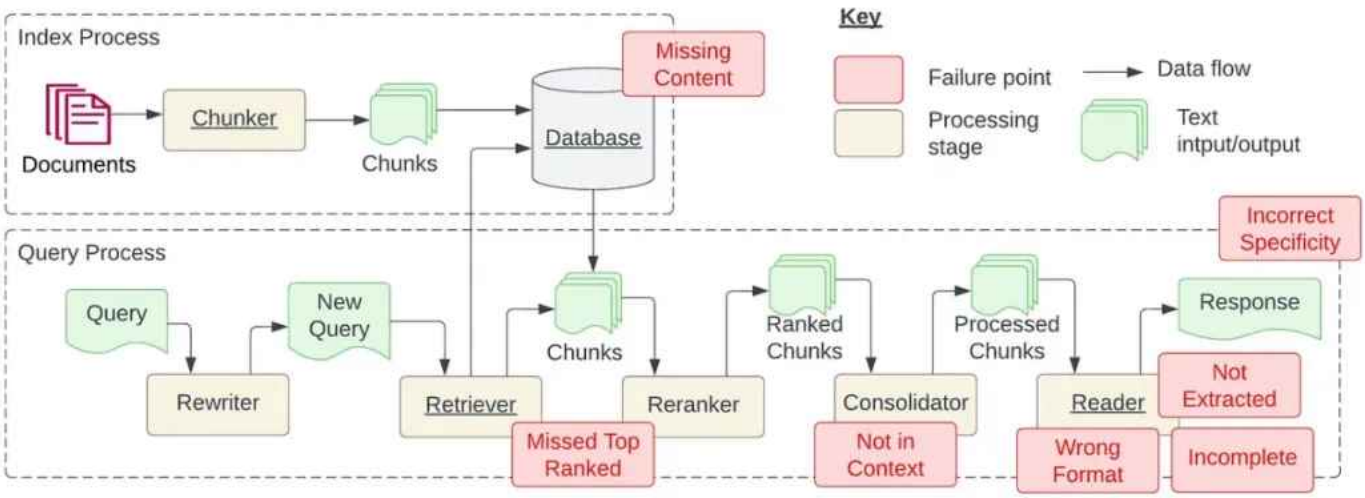


Figure 1: Indexing and Query processes required for creating a Retrieval Augmented Generation (RAG) system. The indexing process is typically done at development time and queries at runtime. Failure points identified in this study are shown in red boxes. All required stages are underlined. Figure expanded from [19].

1. **内容缺失**：知识库中缺失上下文，rag只能提供不精确的答案。

- 解决方案：可以通过数据清洗和prompt优化（比如在prompt中加入“如果你不确定答案是什么，就告诉我你不知道”的提示，防止模型胡说八道）缓解。

2. **错过排名靠前的文档**：由于检索时缺乏上下文，导致检索到的关键的文档排名靠后，没有返回给用户。

- 解决方案：可以通过chunk_size 和 similarity_top_k平衡计算效率和质量，以及采用重新排名的算法。

3. **不在上下文中**：无法将检索到的文档包含在生成答案的上下文中，尤其是检索到大量文档时。

- 解决方案：需要对文档进行合并。

4. **未提取**：LLM倾向于检索近似值而不是精确值，导致包含很多不相关的甚至互相矛盾的信息，可能因为这些噪音损害响应质量。

- 解决方案：数据清洗，压缩prompt，避免中部丢失问题（模型对输入上下文开头和结尾的信息理解能力更强，应避免将关进啊嘻嘻放在中部）。

5. **格式错误**： LLM 忽视了提取特定格式的信息（如表格或列表）的指令。

- 解决方案：prompt中给出示例，简化清晰prompt，对输出进行解析。

6. **特异度不正确**：输出的粒度与输入不一致。

- 解决方案：改进检索策略，如从小到大检索、句子窗口检索、递归检索。

7. **不完整**：输出只回答了输入的部分问题。

- 解决方案：之前RAG的文章中提到的一些方法，如routing到最相关的document store，query重写和分解成子问题。

GraphRAG原理

GraphRAG, 该框架旨在利用大型语言模型（LLMs）从非结构化文本中提取结构化数据, 构建具有标签的知识图谱，以支持数据集问题生成、摘要问答等多种应用场景。 GraphRAG 的一大特色是利用图机器学习算法针对数据集进行语义聚合和层次化分析，因而可以回答一些相对高层级的抽象或总结性问题, 这一点恰好是常规 RAG 系统的短板(例如：用户提问一个问题，需要全局搜索整个数据集，而不是搜索相似性片段，在这种场景下rag性能比较差)。

训练分为两个过程：

Index：

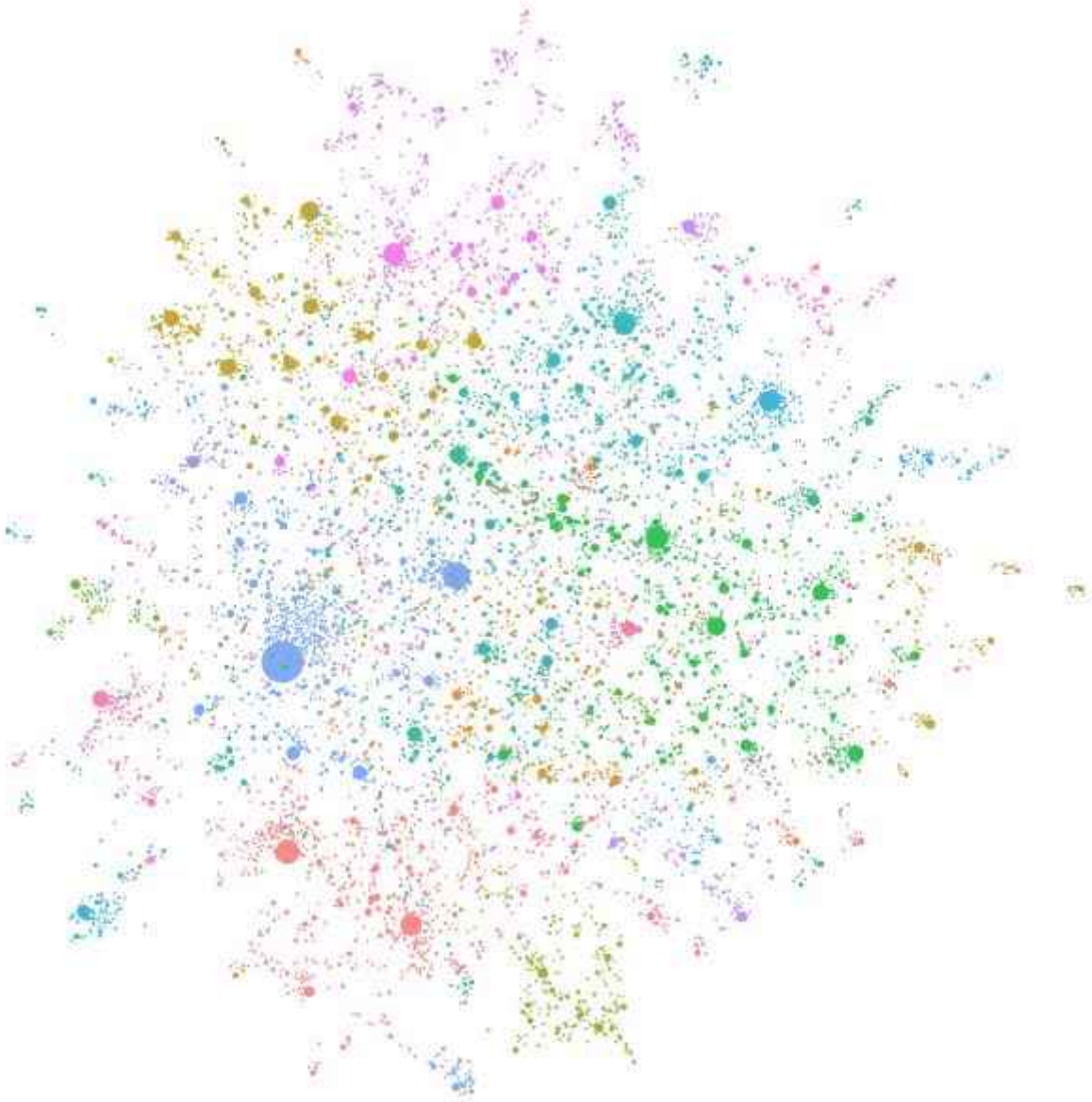
- 将输入语料库切分为一系列文本单元，利用LLM创建对源数据中所有实体和关系的引用，然后使用这些引用来创建 LLM 生成的知识图谱。
- 该图用于创建自下而上的聚类，该聚类将数据分层组织到语义聚类中（在下面的图中使用颜色表示）。
- 自下而上地生成每个组及其组成部分的摘要，这有助于全面理解数据集。

Query：

在查询时，这些结构用于在回答问题时为 LLM 上下文窗口提供材料。采用两种查询模式：

- 全局搜索：利用组摘要，针对有关语料库的整体问题进行推理
- 局部搜索：通过它们的邻居和相关概念，来推理特定实体的本地搜索。

下图中，每个圆圈表示一个实体，实体大小表示实体边的数量，颜色表示实体的分组。颜色分区是一种建立在图形结构之上的自下而上的聚类方法，它使我们能够回答不同抽象级别的问题。



GraphRAG pipeline

参考：深入解析 Graph RAG：提升语言模型问答能力的创新策略

1. **源文档 → 文本块**

- 粒度：将源文档的输入文本分割成块。
- 权衡：更长的块需要更少的 LLM 调用，但可能因为较长的上下文窗口而降低召回率。
- 示例：在 HotPotQA 数据集上，600 token的块大小提取的实体引用几乎是 2400 token块大小的两倍。

2. **文本块 → 元素实例**

- 目标：从文本块中识别和提取图节点和边。使用 LLM 提示识别实体和关系，输出限定元组。
- 定制化：可以通过为LLM提供少量领域特定的示例来定制提示,以适应不同的知识领域(如科学、医学、法律等)。
- 协变量提取:支持二级提取提示,用于提取与提取的节点实例相关的其他协变量,如实体相关的声明、主题、对象、类型、描述、源文本范围以及开始和结束日期。

- 多轮提取：在不牺牲块大小的情况下检测到更多实体。

3. **元素实例 → 元素摘要**

- 摘要：LLM 抽象并总结文本中的实体、关系和声明。
- 处理重复：LLM可能无法始终以统一的格式提取同一实体的引用,可能会产生重复的实体元素和节点，可能会产生重复的实体元素和节点。但由于检测到密切相关的实体及其摘要，加上LLM可以理解多种名称变体对应的共同实体,只要这些变体与一组密切相关的实体有足够的连接性,整体方法就能够应对这种变体。

4. **元素摘要 → 图社区**

- 图建模：创建一个无向加权图，其中节点是实体，边是关系。
- 社区检测：使用 Leiden 算法将图分割成层次化社区，将具有较强内部连接的节点划分为社区,而与图中其他节点的连接较弱。以实现高效的全局摘要。

5. **图社区 → 社区摘要**

- 摘要创建：为每个社区生成报告式摘要。
- 实用性：摘要有助于理解数据集的全局结构和语义，辅助回答全局查询。用户可以浏览不同层级的社区摘要,寻找感兴趣的一般主题,然后深入到较低层级的报告以获取更多细节。

6. **社区摘要 → 社区回答 → 全局回答**

- 准备社区摘要：社区摘要被随机打乱并划分为预设大小的块。这确保相关信息分布在各个块中,而不是集中(并可能丢失)在单个上下文窗口中。
- 社区回答映射：利用社区摘要并行生成社区回答，利用LLM 生成有用程度分数。
- 归纳全局回答：按照有用程度得分降序对中间社区答案进行排序,并迭代地添加到新的上下文窗口中,直到达到token限制。

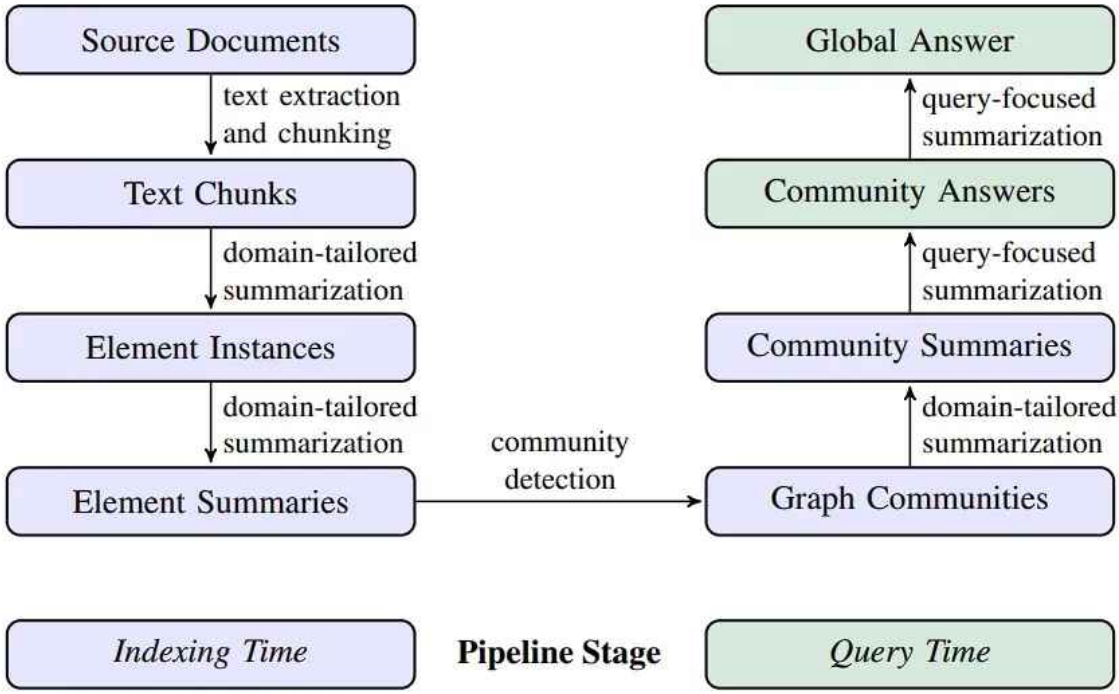


Figure 1: Graph RAG pipeline using an LLM-derived graph index of source document text. This index spans nodes (e.g., entities), edges (e.g., relationships), and covariates (e.g., claims) that have been detected, extracted, and summarized by LLM prompts tailored to the domain of the dataset. Community detection (e.g., Leiden, Traag et al., 2019) is used to partition the graph index into groups of elements (nodes, edges, covariates) that the LLM can summarize in parallel at both indexing time and query time. The "global answer" to a given query is produced using a final round of query-focused summarization over all community summaries reporting relevance to that query.