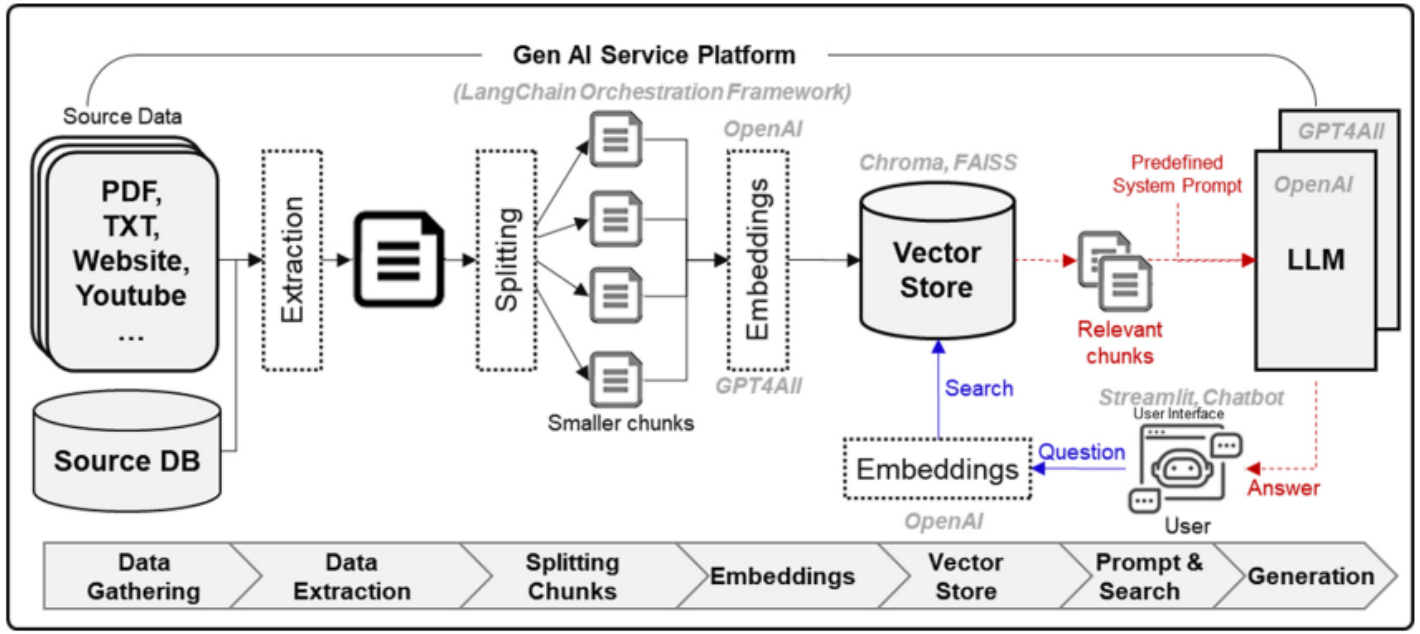
RAG
(Retrieval-Augmented Generation)

jh-cho

RAG(Retrieval-Augmented Generation)

- RAG 모델은 텍스트 생성 작업을 수행하는 모델로 **주어진 소스 데이터로부터 정보를 검색**하고, 해당 정보를 활용하여 원하는 **텍스트를 생성**하는 과정을 수행



✓ **데이터 처리:** 원본 소스 데이터를 **청크(Chunk)단위**의 작은 조각으로 나눔(splitting) → 텍스트를 숫자로 전환하는 **임베딩(Embedding)** → **벡터 저장소(Vector store)**에 저장

✓ **검색** 및 **생성**: 쿼리가 주어지면, 의미기반 검색 사용 **쿼리 임베딩**을 통해 벡터 DB에서 **청크 검색** → 디코딩하여 생성 과정에 활용 → LLM으로 **텍스트 생성**

RAG(Retrieval-Augmented Generation)

- **“Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”**

  Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, Douwe Kiela

  May **2020**

- 사전 훈련된 모델의 한계점
  - 새로운 지식을 업데이트하거나, 기존 지식을 수정하기 어려움
  - hallucination(사실이 아닌 것을 사실처럼 말하는 경우) 문제가 있음
- parametric memory와 non-parametric (retrieval-based) memory를 결합한 하이브리드 모델들
  - (당시 기준) REALM, ORQA는 encoder-only 구조로 이루어졌기 때문에 open-domain extractive QA에 대한 가능성만 탐구
  - 이 논문에서는 이런 hybrid 접근법에 seq-seq 구조를 도입해 좀 더 넓은 가능성을 탐구
- 논문에서 제안하는 모델
  - parametric memory: BART
  - non-parametric memory(document index): dense vector of Wikipedia
  - retriever: DPR(Dense Passage Retriever)

RAG(Retrieval-Augmented Generation)

- Notations
  - x: input sequence
  - z: text document to retrieve
  - y: target sequence to generate
  - x가 들어왔을 때, y를 생성하기 위해 z를 참고

- Components
  - $p_\eta(z|x)$: retriever
    - η: parameter
    - 쿼리 x가 주어졌을 때 passage들에 대한 distribution을 리턴
    - DPR 사용
  - $p_\theta(y_i|x, z, y_{1:i-1})$: generator
    - θ: parameter
    - 인풋 x와 검색된 passage z, 이전의 $i-1$개의 토큰을 기반으로 다음 토큰 생성
    - BART 사용 (단, 어떠한 encoder-decoder든 사용가능)
  - 이 retriever와 generator는 훈련 과정에서 동시에 학습

RAG(Retrieval-Augmented Generation)

- Retriever: DPR(Dense Passage Retrieval, 2019)

$$p_\eta(z|x) \propto \exp\left(\mathbf{d}(z)^\top \mathbf{q}(x)\right) \qquad \mathbf{d}(z) = \mathrm{BERT}_d(z), \;\; \mathbf{q}(x) = \mathrm{BERT}_q(x)$$

  - $d(z)$는 document encoder를 통해 산출되는 dense representation
  - $q(x)$는 query encoder를 통해 산출되는 query representation
  - x에 대한 z의 분포는 $d(z)$와 $q(x)$의 내적 연산을 기반으로 하여 산출
  - 내적 값이 높은 순서대로 top-k document를 골라 검색(retrieve) → MIPS(Maximum Inner Product Search) 사용

- Generator: BART(2019) (단, encoder-decoder 모델이면 다 가능)

$$p_\theta\left(y_i|x, z, y_{1:i-1}\right)$$

  - BART의 encoder, decoder를 사용하여 모델링하며 400M parameter를 가진 BART-large를 사용
  - input x와 검색된 컨텐츠 z를 결합하기 위해 간단하게 concatenation

RAG(Retrieval-Augmented Generation)

- Decoding
  - Output (y)을 산출하기 위해 retrieve된 document에 대해 marginalize*하게 되는데, 저자들은 이 marginalize방식을 다르게 한 두 가지 모델인 **RAG-Sequence**와 **RAG-Token**을 제안
  - * marginalize는 output y에 대한 확률만을 구하기 위해 각 document z에 대해 구해진 곱사건의 확률을 모두 더하는 것

- Models
  - RAG-Sequence model

$$p_{\text{RAG-Sequence}}(y|x) \approx \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) p_\theta(y|x,z) = \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) \prod_i^N p_\theta(y_i|x,z,y_{1:i-1})$$

  - RAG-Token model

$$p_{\text{RAG-Token}}(y|x) \approx \prod_i^N \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) p_\theta(y_i|x,z,y_{1:i-1})$$

  - RAG-Sequence는 document에 대한 값을 sequence 단위로 고려한 다음 marginalize. 즉, 완성된 sequence를 생성하기 위해 같은 문서만을 사용
  - RAG-Token은 document에 대한 값을 token단위로 고려한 다음 marginalize 하고, 다음 token을 생성하면서 sequence를 생성. 즉, 각 타겟 토큰마다 다른 문서를 사용

RAG(Retrieval-Augmented Generation)

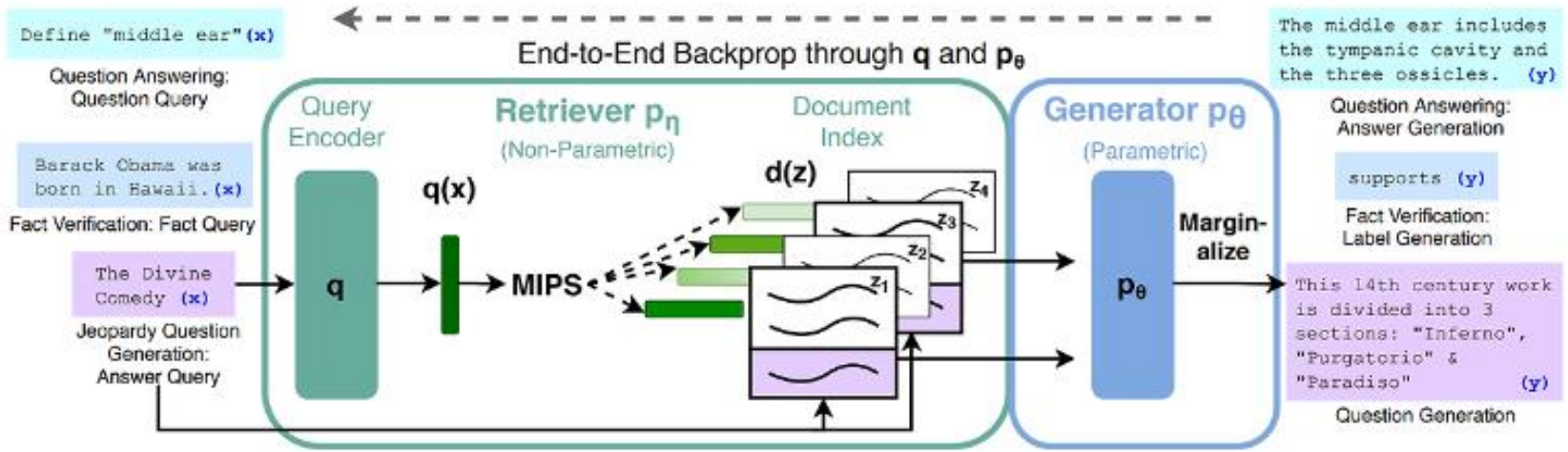
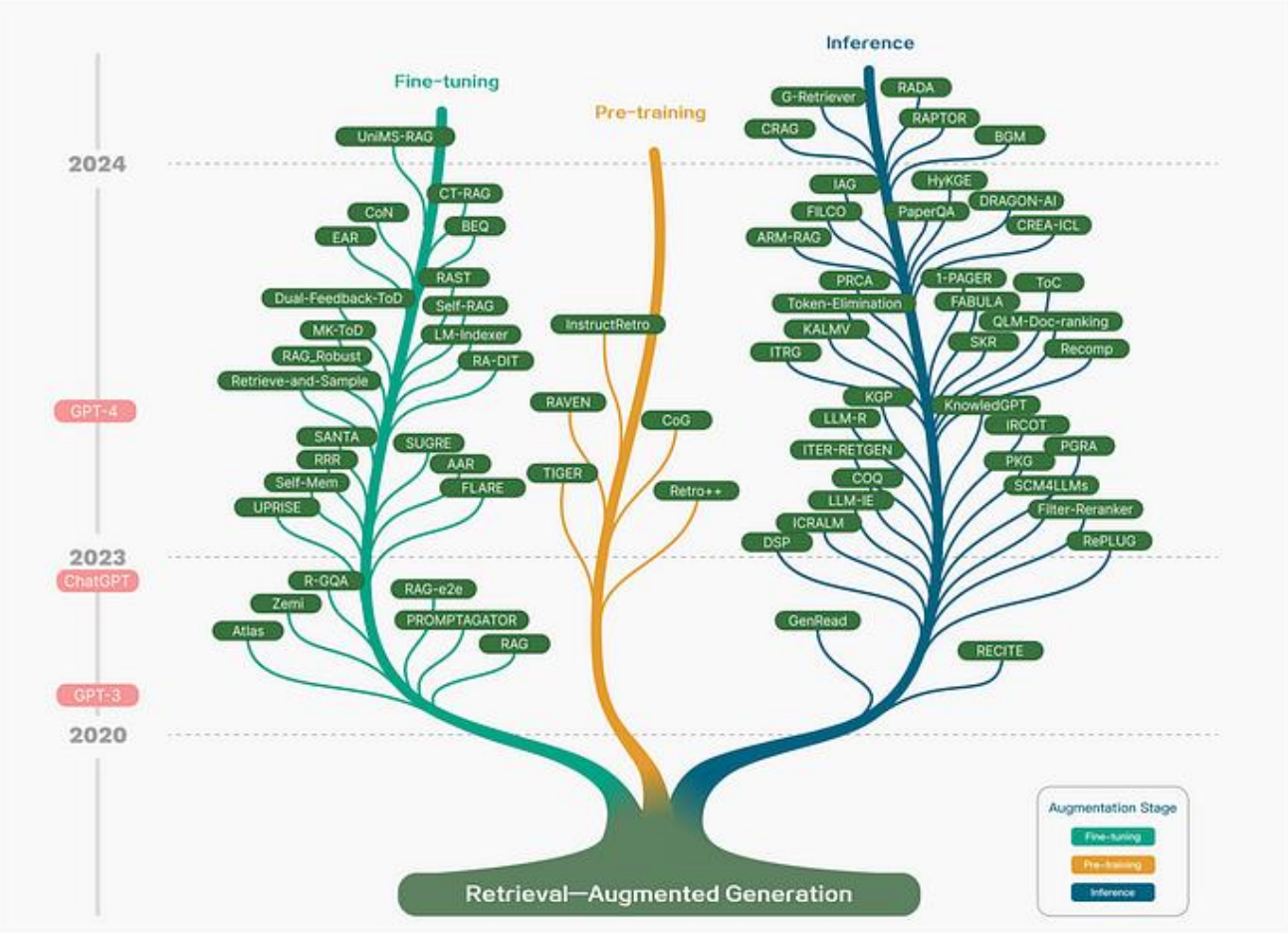


Figure 1: Overview of our approach. We combine a pre-trained retriever (*Query Encoder* + *Document Index*) with a pre-trained seq2seq model (*Generator*) and fine-tune end-to-end. For query x, we use Maximum Inner Product Search (MIPS) to find the top-K documents z_i. For final prediction y, we treat z as a latent variable and marginalize over seq2seq predictions given different documents.

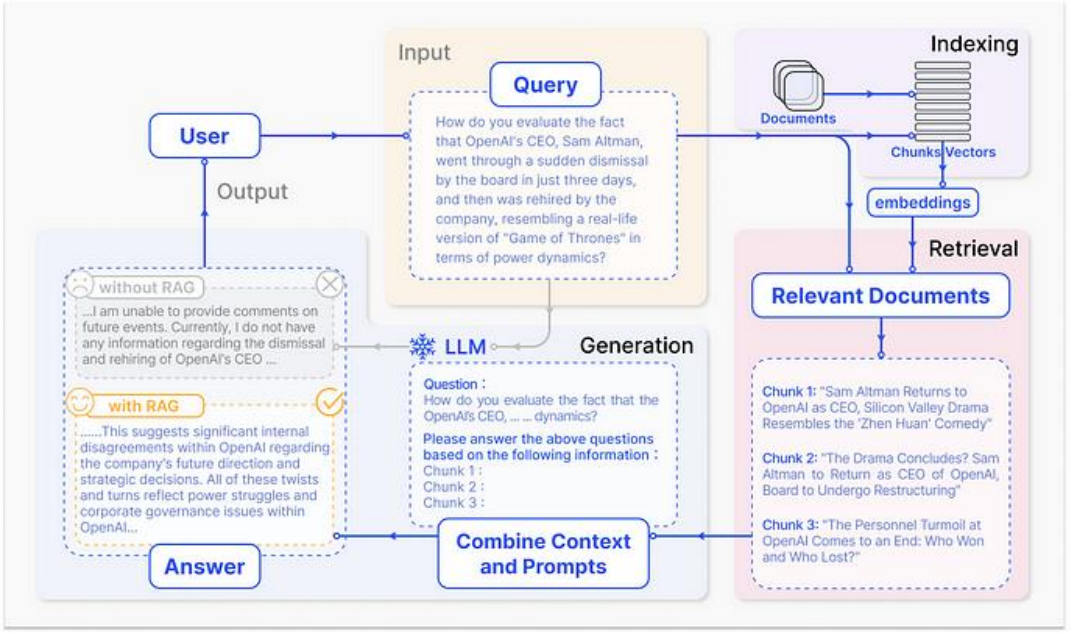
RAG(Retrieval-Augmented Generation)

- **“Retrieval-augmented generation for large language models: A survey”**
  Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang
  Dec **2023**



RAG(Retrieval-Augmented Generation)

- **“Retrieval-augmented generation for large language models: A survey”**
  Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang
  Dec **2023**



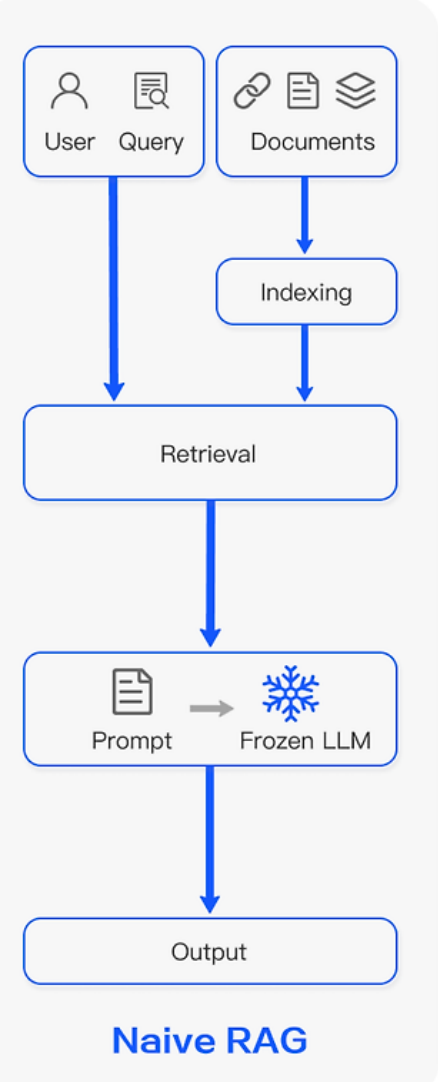
1) Indexing: 문서들을 청크로 분리 및 벡터로 인코딩하여 벡터 데이터베이스에 저장
2) Retrieval: Query와 관련된 Top K개의 chunk를 가져옴
3) Generation: 원본 Query와 Retrieval로 가져온 chunk들을 LLM에 input으로 넣어 최종 답변 생성

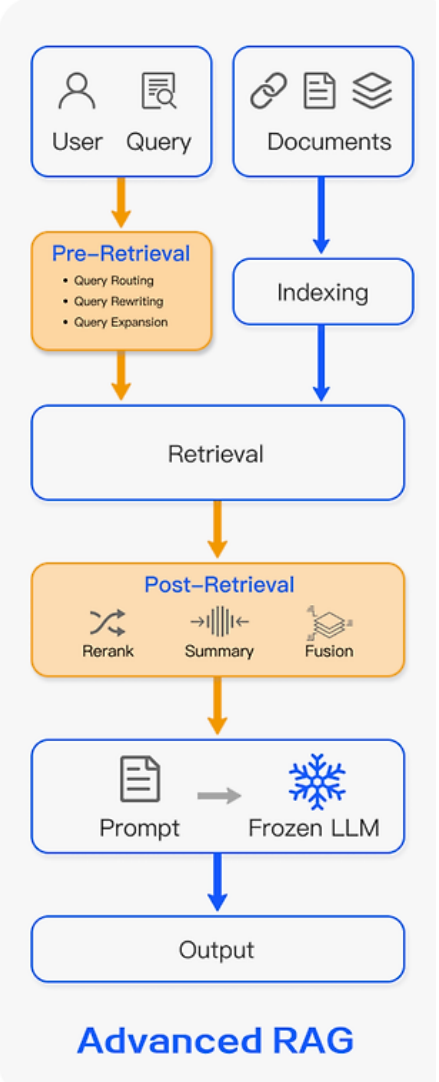
RAG(Retrieval-Augmented Generation)

- RAG Framework
  - 초기 RAG인 Naive-RAG → Advanced-RAG → Modular-RAG 등장
- Naïve-RAG
  - Indexing
    - PDF, HTML, Docx, 마크다운 등에서 raw data 추출 → standard text
    - Text → chunk size로 split → embedding → vector DB
  - Retrieval
    - 유저의 쿼리를 embedding 모델을 사용해 비교
    - Similarity score를 통해 가장 유사한 top K개 chunk를 검색
  - Generation
    - 쿼리와 검색한 정보를 종합해 언어모델로 답변 생성
  - 단점
    - Retrieval Challenges: Retrieval 시 정확도(precision) 및 재현율(recall)이 좋지 않아 중요 정보를 빠뜨리는 경우
    - Generation Difficulties: Hallucination 이슈
    - Augmentation Hurdles: 문맥이 일치하지 않거나 아예 일관성이 없는 정보가 나올 수도 있r고 중복이 나올 수 있음
    - → 단점을 보완한 Advanced RAG 등장

RAG(Retrieval-Augmented Generation)

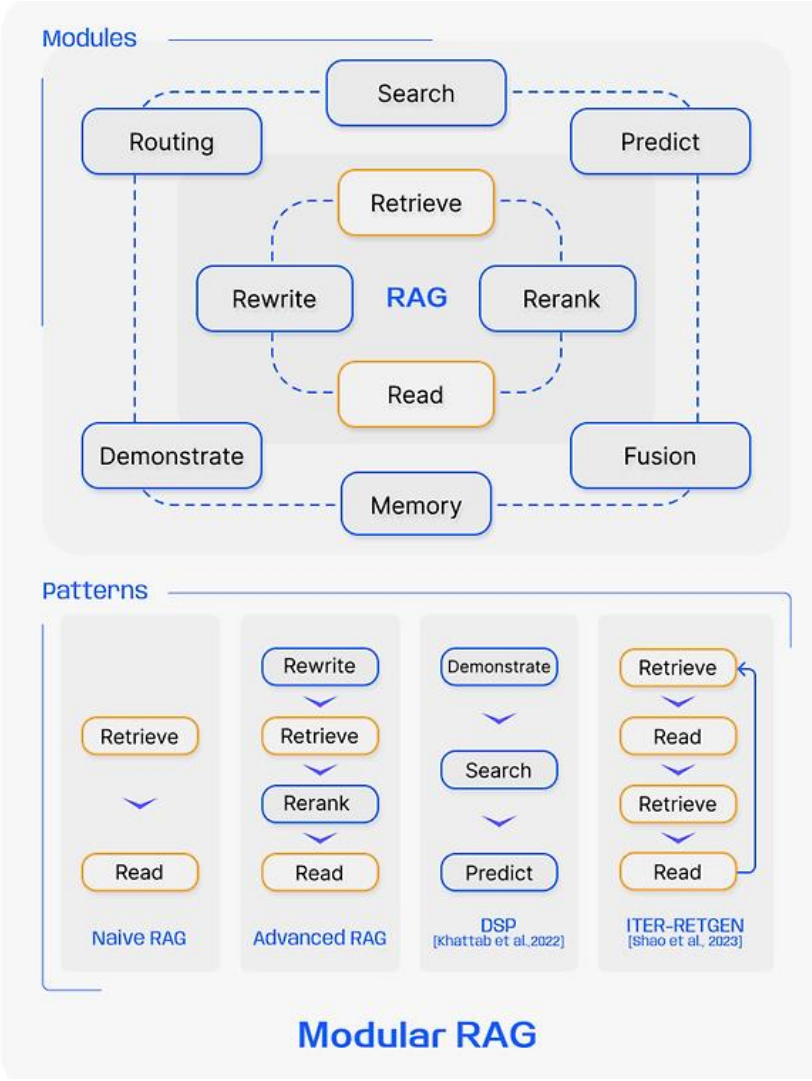
- Advanced-RAG
  - 검색과정 전후로 기존 RAG의 단점을 보완하는 과정 추가
  - Pre-Retrieval
    - Index optimization:
      - 인덱싱된 컨텐츠의 질 향상
      - 방법: enhancing data granularity, optimizing index structures, adding metadata, alignment optimization, and mixed retrieval
    - Query optimization
      - 유저의 원래 질문을 명확히 하고, 검색에 알맞게 함
      - 방법: query rewriting, query transformation, query expansion and others
  - Post-Retrieval
    - Rerank chunks
      - 검색된 정보 순위를 다시 지정하여 관련성 높은 컨텐츠 재배치
    - Context compressing
      - 모든 관련 문서 입력 → 정보과부하 / 따라서 필수 정보 선택, 중요 섹션 강조, 문맥 단축 등 활용



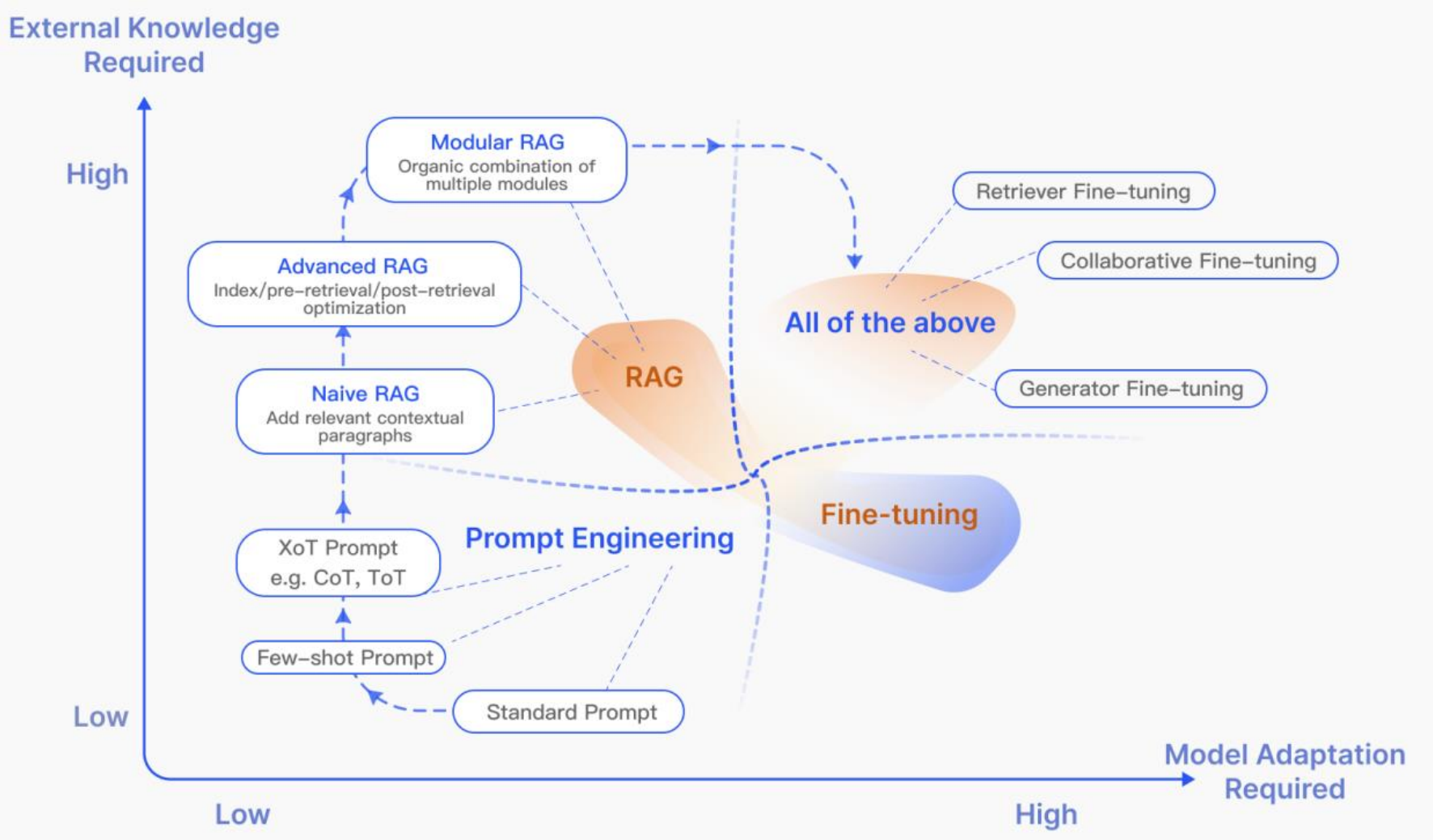


RAG(Retrieval-Augmented Generation)

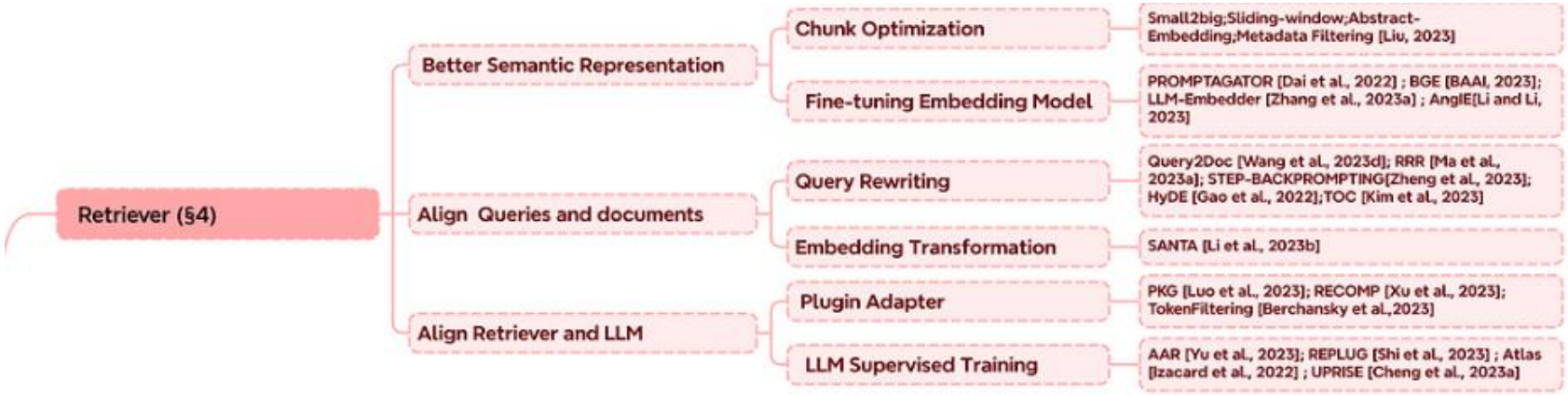
- Modular RAG
  - Modules
    - **Search module**:
      - 기존은 embedding vector을 바탕으로 유사성을 판단하였다면, Search Module은 검색 엔진, 지식 그래프, 테이블 데이터, 텍스트 데이터를 이용하여 직접 찾는 방식
    - **Routing**
      - query를 분석하여 최적의 data 형태 선택, 인덱스 범위 제한
    - **Memory module**:
      - 메모리를 사용하여 가장 유사한 질문 기억과 비교 & 반복 학습
    - **Fusion**
      - 기존 query를 LLM을 이용해 다중 query로 확장, re-rank
  - Patterns
    - 특정 문제를 해결하기 위해 모듈을 대체하거나 재조정. 즉, 자유롭게 형태 변경



RAG(Retrieval-Augmented Generation)

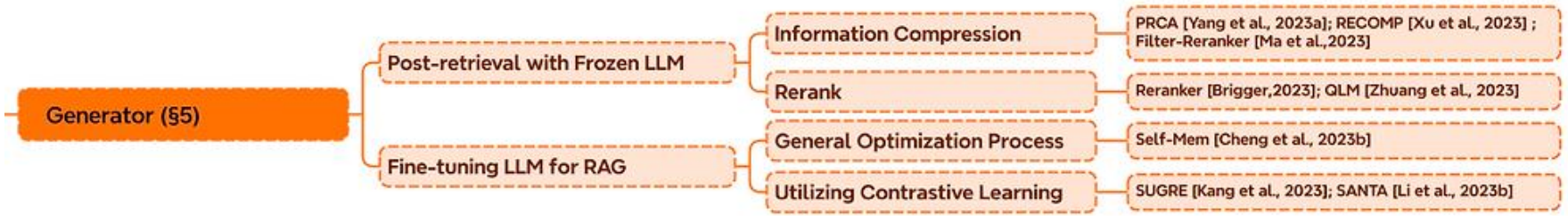


RAG(Retrieval-Augmented Generation)



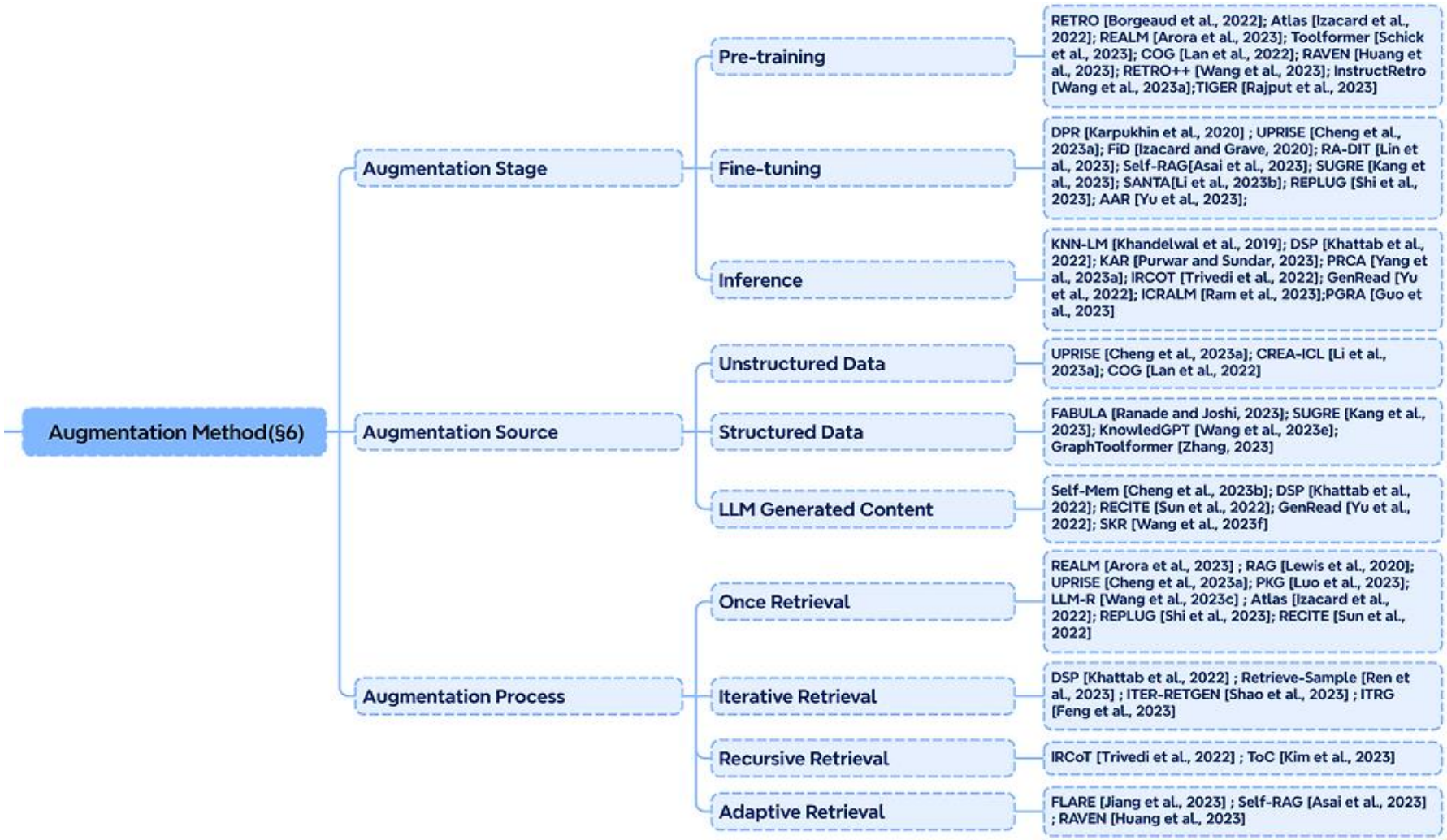
- Semantic Representation
  - **Chunk Optimization: Chunk의 사이즈**를 어느 정도로 해야 할지 결정, 각 모델이나 목적에 맞게 설정
    - Sliding window, small2big, meta-data filtering, graph index technique 등 기술 사용
  - **Fine-tuning Embedding Model:** 모델이 chunk를 잘 이해(& embdedding)할 수 있도록 임베딩 모델을 파인튜닝
- Aligning Queries and Documents
  - **Query rewriting**: query와 document의 의미 재정렬
  - Embedding transformation: Query를 rewriting하는 것이 아닌, embedding값 자체를 더욱 정확하게 제시
- Align Retriever and LLM
  - Plugin Adapter
  - LLM Supervised Training

RAG(Retrieval-Augmented Generation)



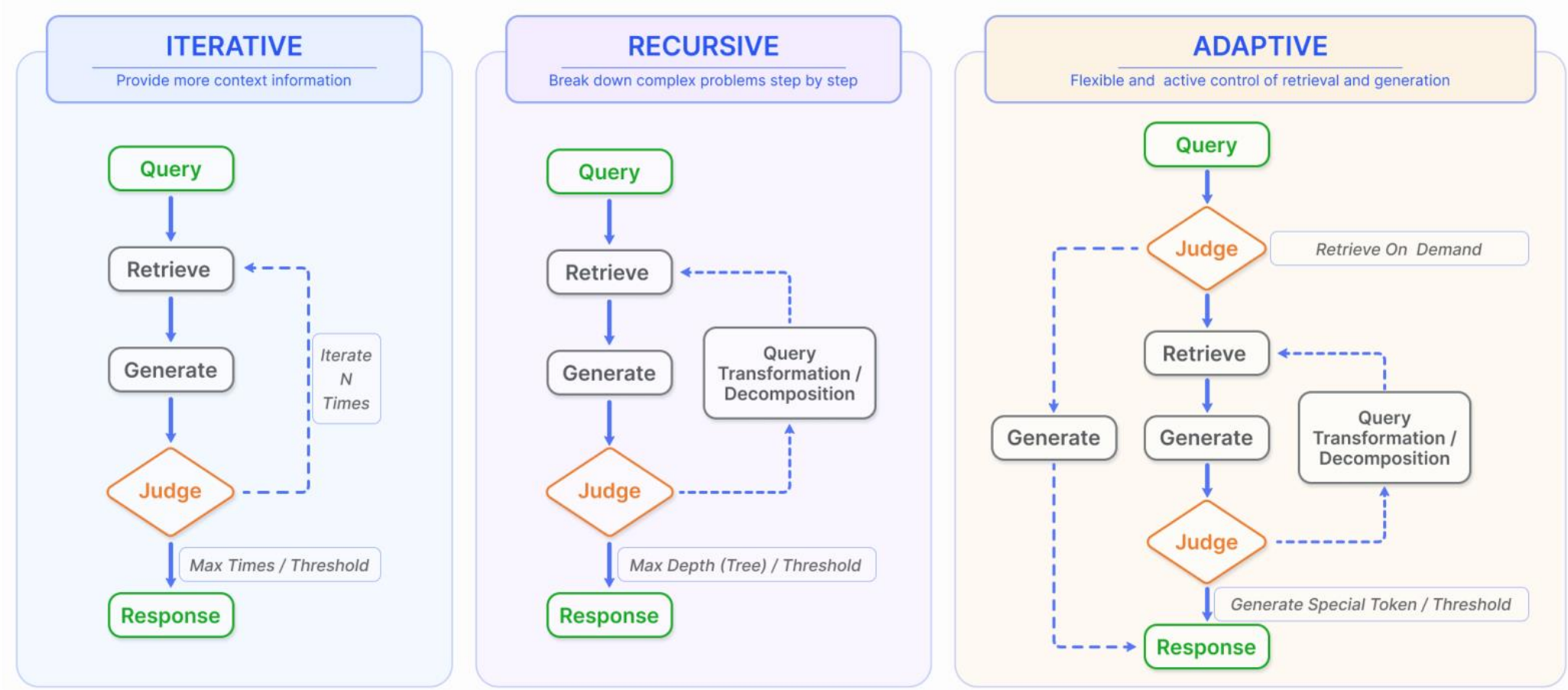
- Post-retrieval with Frozen LLM
  - Information Compression: 검색한 문서에서 꼭 필요한 정보만 정리
  - Rerank
- Fine-tuning LLM for RAG
  - General optimization process
  - Utilizing contrastive learning

RAG(Retrieval-Augmented Generation)

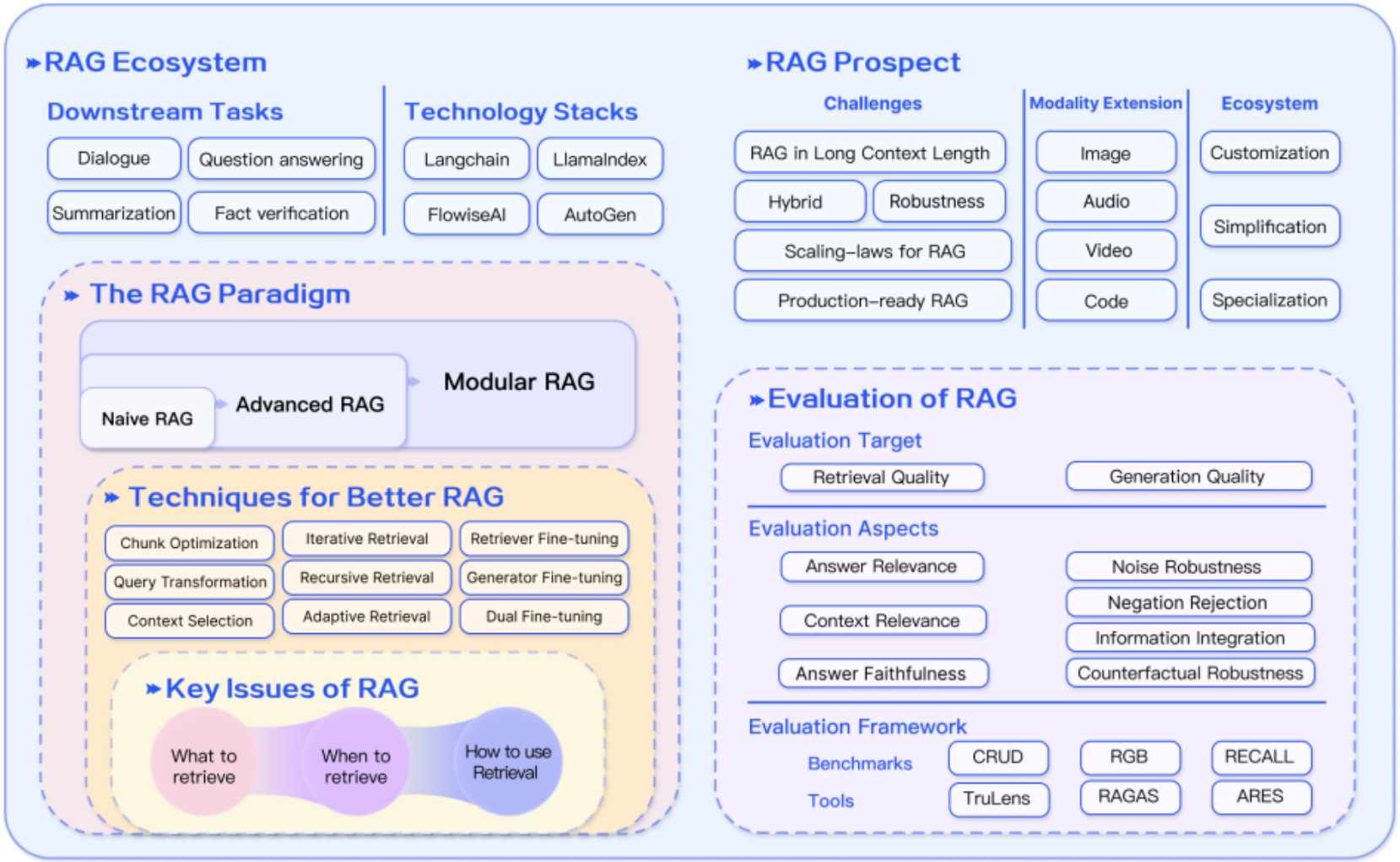


RAG(Retrieval-Augmented Generation)

- Augmentation process



RAG(Retrieval-Augmented Generation)



References

- Gao, Yunfan, et al. "Retrieval-augmented generation for large language models: A survey." arXiv preprint arXiv:2312.10997 (2023).
- Lewis, Patrick, et al. "Retrieval-augmented generation for knowledge-intensive nlp tasks." Advances in Neural Information Processing Systems 33 (2020): 9459-9474.
- 정천수. "LLM 애플리케이션 아키텍처를 활용한 생성형 AI 서비스 구현: RAG모델과 LangChain 프레임워크 기반." 지능정보연구 29.4 (2023): 129-164.