JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 29, No. 8: 1003~1008, Aug. 2025

Agentic RAG 시스템 구현 경험에 기반한 설계 인사이트

김재호[1] · 김장영[2*]

Design Insights from Empirical Experiences in Implementing Agentic RAG Systems

Jae-Ho Kim[1] · Jang-Young Kim[2*]

[1]Graduate Student, Department of Computer Science, The University of Suwon, Hwaseong, 18323 Korea
[2*]Associate Professor, Department of Computer Science, The University of Suwon, Hwaseong, 18323 Korea

요 약

Retrieval-Augmented Generation(RAG)과 Fine-tuning 기반 Slim Language Model(SLM)의 구조적 차이와 장단점, 그리고 Agentic RAG 시스템 설계 시 고려해야 할 핵심 요소를 분석하였다. 텍스트 분할 방법, 임베딩 모델 선정, LLM 및 에이전트 워크플로우의 아키텍처 선택이 전체 시스템 성능, 검색 정확도, 안정성에 미치는 영향을 살펴보았다. 또한, 무한루프, 분기 조건의 모호성, 도구 선택의 불일치 등 실제로 발생할 수 있는 문제점과 그 해결 방안도 제시하였다. 연구 결과, 자동 오류 감지 및 회복 로직의 구현, 도메인 특화 워크플로우 최적화, LLM과 임베딩 모델의 정량적 평가가 중요함을 확인하였다. 더불어 실시간 사용자 피드백 반영이 시스템의 적응성과 동적 관리에 효과적임을 제안한다. 이러한 방향들은 RAG 및 Agentic RAG 시스템의 신뢰성, 확장성, 실제 적용 가능성 향상에 중요한 연구 과제가 될 것이다.

ABSTRACT

The structural differences, strengths, and limitations of Retrieval-Augmented Generation (RAG) compared to fine-tuned Slim Language Models (SLMs) are examined, focusing on key considerations for designing Agentic RAG systems. The analysis explores how choices in text chunking methods, embedding model selection, and the architecture of LLMs and agentic workflows impact overall system performance, retrieval precision, and operational stability. Challenges such as infinite loops, ambiguous branching, and inconsistent tool selection are discussed, along with potential strategies for mitigation. The findings highlight the importance of implementing automatic error detection and recovery, optimizing domain-specific workflows, and quantitatively evaluating both LLMs and embedding models. Moreover, incorporating real-time user feedback is identified as an effective approach for improving adaptability and dynamic workflow management. These directions are suggested as important focal points for future research and development, aiming to enhance the reliability, scalability, and practical applicability of RAG and Agentic RAG frameworks in advanced AI applications.

키워드 : 임베딩, 대형 언어 모델, 검색 증강 생성(RAG), 에이전틱 RAG

Keywords : Embedding, LLM(Large Language Model), RAG(Retrieval Augmented Generation), Agentic RAG

Received 8 July 2025, **Revised** 10 July 2025, **Accepted** 28 July 2025

*** Corresponding Author** Jang-Young Kim(E-mail:jykim77@suwon.ac.kr, Tel:+82-31-229-8345)
Associate Professor, Department of Computer Science , The University of Suwon, Hwasung, 18323 Korea

Open Access http://doi.org/10.6109/jkiice.2025.29.8.1003 pISSN:2234-4772

Ⅰ. 서 론

최근 LLM(대형 언어 모델) 분야에서 가장 주목받는 기술로는 RAG(Retrieval-Augmented Generation)와 fine-tuning을 통한 SLM(Slim Language Model)이 꼽힌다. 최근 여러 컨퍼런스 동향을 보면, RAG와 SLM(fine-tuning)이며, 강연 중 절반 이상은 SLM 및 fine-tuning, 모델 아키텍처 관련 논의가 집중적으로 이루어지고 있다.

그러나 최근 MCP 및 AI Agent 기술이 빠르게 발전함에 따라, 실제 현업 및 서비스 환경에서는 RAG의 중요성이 다시금 부각되고 있다고 필자는 판단한다. 본 논문은 이러한 최신 기술 동향을 고려할 때, 기존 fine-tuning 중심의 접근법만으로는 한계가 존재하며, RAG가 향후 LLM 기반 AI 시스템에서 더욱 전략적이고 핵심적인 역할을 할 것으로 기대한다[1]. 이에 본 연구에서는 RAG를 중심으로 한 경험적 분석과 실제 적용 사례를 제시하고, AI Agent 환경에서의 RAG 설계 및 운용상의 인사이트를 제공하고자 한다.

본 논문에서는 RAG 시스템을 구현하면서 경험한 다양한 사례를 바탕으로, RAG 구조와 Agentic 환경에서의 LLM 및 Embedding 모델 선택이 시스템 성능에 미치는 영향, 텍스트 분할 및 노드 구조에 따른 출력 방식과 무한루프 발생 가능성등을 고찰한다.이를 통해 기존 연구에서 다루지 않은 실제적인 RAG 설계 및 운용 인사이트를 제시하고자 한다.

1.1 Fine-Tuning과 RAG

Fine-tuning은 특정 작업이나 도메인에 맞춰 LLM의 파라미터를 재학습시켜, 해당 지식을 모델 내부에 내재화하는 방식이다. 이는 특정 도메인 지식에 특화된 경량 모델 구현, 프롬프트 단순화 등의 이점을 제공하나, 대규모 연산 자원, 고품질 데이터, 장시간의 학습이 필요하며 오버피팅, 데이터 편향, 파라미터 충돌 등의 문제가 발생할 수 있다[2].

반면, RAG는 외부 문서 기반 검색을 통해 LLM이 내재하지 않은 지식을 실시간으로 보완하는 구조이다. RAG는 실시간성, 도메인 유연성, 최신 정보 반영 등에서 강점을 가지며, 데이터 품질 부담이 상대적으로 낮고, 할루시네이션 현상 완화 및 재학습 불필요성 등의 장점이 있다. 단, 검색 및 임베딩 성능에 따라 답변의 품질이 좌우되며, 복잡한 질문이나 맥락에서는 성능 저하가 발생할 수 있다[3].

AI 에이전트 기반 modular RAG가 등장하여, 다양한 도구 및 메모리, 사전 지식을 바탕으로 복합적인 검색 및 추론을 수행하는 구조가 제안되고 있다. 이러한 구조에서는 각 에이전트가 도메인별 데이터 검색 및 답변 생성을 담당하며, fine-tuning이 제공하는 지식 내재화 효과를 일정 부분 대체할 수 있음을 시사한다[4].

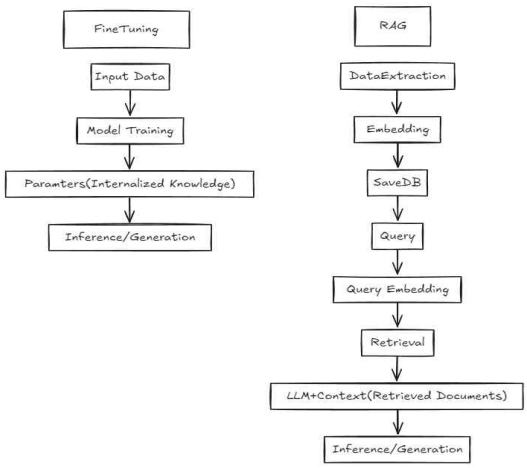

Fig. 1 Workflow Comparison of Fine-Tuning and RAG

위 그림 1은, Fine-tining과 RAG의 흐름을 표로 간략히 비교하는 이미지다.

Ⅱ. 배경 및 기존연구

최근 LLM 기반 생성형 AI 분야에서는, 외부 지식의 효과적인 활용을 위해 RAG와 fine-tuning을 통한 SLM이 주요 연구 주제로 부각되고 있다. 특히 RAG는 LLM의 한계로 지적되는 최신성, 도메인 특화, 할루시네이션 완화 등에 효과적이라는 점에서 다양한 실무 및 연구 환경에서 빠르게 도입되고 있다. Lewis et al. (2020)은 외부 문서로부터 정보를 검색하고, 이를 기반으로 LLM이 응답을 생성하는 RAG 구조를 최초로 제안하였다[3]. 이 방식은 LLM의 내재 지식 한계와 데이터 최신성 문제를 효과적으로 보완한다.

그러나 기존 RAG는 주로 정적 파이프라인 구조로 설

계되어, 복잡한 멀티스텝 추론, 동적 분기, 도구 호출 등에서는 유연성이 떨어진다는 한계가 지적된다. 이러한 한계를 극복하기 위해, reflection, planning, tool-use, multi-agent collaboration 등 에이전트 기반의 패턴을 도입한 Agentic RAG 개념이 대두되고 있다. 기존 RAG의 단점을 보완하고, 다양한 워크플로우 설계 및 도구 활용이 가능한 Agentic RAG의 개념과 발전 방향을 체계적으로 정리하였다. 이 연구에서는 Single Agent Router, Multi Agent System, Hierarchical/ Graph-Based 등 다양한 Agentic RAG 구조와 실제 적용 사례, 운영상의 도전 과제까지 종합적으로 다루고 있다[4].

본 연구는 기존 RAG와 Agentic RAG의 주요 연구 흐름을 바탕으로, RAG 시스템 설계, 에이전트/워크플로우 도입, 그리고 실제 적용 과정에서 확인된 개선 방향에 대해 고찰한다.

Ⅲ. 운영 경험과 주요 쟁점

4.1 Text_splitter 및 Embedding 모델

RAG 시스템에서 대표적으로 사용하는 텍스트 분할 방법에는 Recursive Character Text Splitter와 Semantic Chunker가 있다[5]. Recursive Character Text Splitter는 일정 길이로 텍스트를 단순하게 나누는 방식이고, Semantic Chunker는 문서의 의미적 경계를 기준으로 분할하는 방식이다.

임베딩 모델은, Vector DB에 저장할 벡터를 생성하거나, Semantic Chunker에서 의미 단위 분할 시에 사용된다. 임베딩 모델은 벡터 차원, 도메인 적합도 등에 따라 성능이 달라지며, BGE-M3와 OpenAI 모델이 대표적이다.

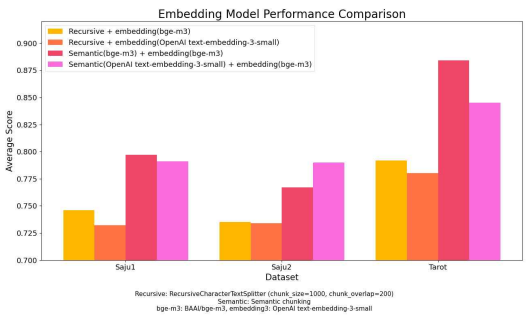

Fig. 2 Comparison of Embedding Model and Text Splitter Performance

위 그림2 에서는, 검색 성능을 비교를 보여준다. 실제 프로젝트에서 사용했던 3종의 영문 서적 텍스트(사주, 타로)를 활용하였다. 이 데이터들을 일반적으로 좋은 성능을 가지는, RecursiveCharacter TextSplitter (chunk_size=1000, chunk_overlap=200)로 분할하였고 [6], Semantic Chinker의 엠베딩은 BGE-M3, text embedding 3 small 모델을 사용하였다. 벡터 데이테 베이스 임베딩에 있어서는, BGE-M3와 text embedding 3 small 임베딩 모델을 각각 적용하여, 각 설정이 RAG의 검색 정확도와 인덱싱 효율에 미치는 차이를 경험적으로 관찰할 수 있었다. 임베딩 모델은 학습 데이터의 도메인에 따라 실제 검색 정확도 및 표현력에서 차이가 발생할 수 있다.

4.2 LLM 선택에 따른 경험

LLM은 Open Source LLM(Llama, Deepseek, Gemma 등)과 Closed Source LLM(ChatGPT, Claude, Gemini 등)으로 구분될 수 있다. 많은 사용자들은 비용 및 보안성 때문에 OpenSource LLM을 많이 선호하지만, 고성능 GPU와 속도 한계등 현실적인 제약으로 인해, 실제 서비스나 실험에서는 API기반인 Closed LLM을 많이 사용하게 되는 경우가 발생한다.. 특히 AI Agent에 있어서, Closed LLM은 fine-tuning된 Instuction model이 필요한데 낮은 Intelligence 수준이 낮은 경우 성능 저하가 뚜렷하게 나타났다[7].

본 연구에서는 대표적 Closed LLM인 GPT 계열 모델을 대상으로, 프로젝트 환경에서 retriever(문서 검색), prompt 이해, Agentic RAG(에이전트적 워크플로우) 수행 능력을 비교하였다.

Table. 1 Comparative Performance of Different GPT Models

Model	Retriever	Prompt Understanding	Agentic RAG Capability
gpt-4o-mini	Fair	Fair	Poor
gpt-4o	Excellent	Excellent	Excellent
gpt-4.1-mini	Excellent	Excellent	Excellent
gpt-4.1-nano	Poor	Poor	Poor

위 표 1는 gpt모델별 수행력 평가이다. 실험 결과 gpt-4o-mini는 프롬프트 해석에 다소 한계가 있었고,

retriever에서도 일부 제약이 있었다. 반면, gpt-4o와 gpt-4.1-mini는 retriever, prompt 해석, agentic RAG 모두에서 우수한 성능을 보였다. gpt-4.1-nano는 prompt 이해와 agentic rag 수행 모두에서 성능이 미흡했으며,retriever 자체도 정확하게 동작하지 못했다.

4.3 AI Agent에서의 경험

최근 대형 언어모델(LLM)의 발전과 더불어, 복잡한 작업을 자동화할 수 있는 AI Agent 시스템의 필요성이 크게 부각되고 있다. 특히, 문서 기반 답변에서 RAG가 등장하면서, 단순한 답변 생성에서 벗어나 문맥 파악, 동적 추론, 그리고 다양한 도구 호출 등 복합적인 워크플로우를 스스로 설계하고 실행할 수 있는 Agentic RAG 구조가 주목받고 있다. 이러한 시스템은 사용자의 질문 의도와 문맥을 파악한 뒤, 필요한 도구를 선택해 호출하거나, 복수의 에이전트가 협력하여 문제를 해결하는 Multi-Agent System으로 확장될 수 있다는 점에서 기존 RAG와 구별된다. 본 논문에서는 이처럼 동적인 워크플로우 설계, 문맥 기반 의사결정, 자동 도구 활용이 가능해진 Agentic RAG 구현 경험과, 실제 운용 과정에서 얻은 실무적 인사이트를 중심으로 각 툴과 시스템의 특징을 분석하였다.

AI Agent 기반 RAG 시스템의 설계에서는 많은 개발자와 연구자들이 다양한 선택지 앞에서 고민하게 된다. 특히, Single Agent System과 Multi Agent System 중 어떤 구조를 채택할지, 그리고 에이전트와 툴을 연결하는 올바른 구현 방식에 대한 고민이 깊다. Agent와 tool을 연결하는 방식에서, LangChain과 LangGraph를 활용하는 실제 개발 환경은 다음과 같다.

첫째, 여러 툴을 리스트로 묶어 llm.bind_tools를 통해 LLM에 직접 바인딩하는 방식이 있다. 이 방법은 코드가 간결하고, 다양한 툴을 손쉽게 실험적으로 추가하거나 제거할 수 있다는 장점이 있다. 그러나 이 방식은 각 툴의 역할과 사용법에 대한 명확한 설명이 LLM에 충분히 제공되지 않는 경우가 많아, LLM의 툴 선택 로직이 일관되지 않고 불안정해지는 문제가 자주 발생한다. 특히, 대규모 시스템이나 멀티툴 환경에서 llm.bind_tools 방식만 사용할 경우 각 툴의 고유 기능이나 목적이 프롬프트에서 충분히 분리·강조되지 않으므로 LLM이 상황에 따라 잘못된 툴을 선택하거나, 동일한 입력에 대해 서로 다른 툴을 사용하는 등 답변 결과의 일관성이 크게 떨어지는 현상이 나타날 수 있다. 따라서, 소규모 실험이나 빠른 프로토타이핑에는 llm.bind_tools가 편리

둘째, 각 툴을 @tool 데코레이터를 사용해 명확한 설명과 함께 선언한 후, 이를 create react agent와 같이 에이전트 생성 시 명시적으로 등록하는 방식이다. 이 방법은 각 툴의 역할과 입력/출력에 대한 설명이 LLM에 명확히 전달되므로, 툴 선택의 일관성과 신뢰성을 높일 수 있다는 장점이 있다. 특히 대규모 시스템이나 복잡한 멀티에이전트 환경에서 효과적이다. 또한, create retriever tool을 활용해 RAG 특화 에이전트 생성 함수가 도입하여, 검색기와 생성기를 결합하거나 분리하는 다양한 워크플로우 설계가 가능해졌다. 이러한 방식은 RAG 시스템의 특성에 맞게 최적화된 에이전트 구성이 가능하도록 해준다.

AI Agent 시스템의 설계와 구현 방식은 프로젝트 목적, 시스템 복잡도, 그리고 툴 설명 및 역할 분배의 명확성에 따라 신중히 선택해야 한다. 각 방식의 장단점과 한계를 충분히 고려하는 것이 필수적이다.

Table. 2 Comparison of Key Agentic Functions/Patterns in LangChain & LangGraph

Aspect	Create React agent	Create Retrievr tool	Llm bind_tools
Role/ Purpose	Create Agent	Add Retriever Tool	Bind LLM with Tools
Scope/ Level	Agent Level	Tool Level	LLM Level
Strength	Scalability, Integration	Easy to Add Retrieval Tool	Simplicity
Ceveat	Complex Setup, Initialization	Requires Create React agent for Use	Not suitable for large scale system

위 표2에서, LangChain 및 LangGraph 기반의 agentic RAG 시스템을 설계할 때, 대표적으로 세 가지 주요 방식을 고려할 수 있다.

첫째, llm.bind_tools는 LLM에 다양한 도구를 직접 바인딩하여, LLM 자체가 마치 에이전트처럼 도구를 호출할 수 있도록 하는 방식이다. 빠른 실험 및 함수 호출 테스트, 코드 간결성 등에서 유리하나, description 정보

가 LLM에 직접적으로 충분히 전달되지 않으면 일관성 있는 행동이 어렵고, 대규모 시스템에는 부적합할 수 있다. LLM자체가 Tool call agent처럼 행동하게 하는 설정이다.

둘째, create_retriever_tool은 검색기를 agent가 사용할 수 있는 개별 도구로 래핑하는 역할을 한다. 이 방식은 RAG 시스템 내에서 검색 기능을 손쉽게 agent에 추가할 수 있다는 이점이 있다. 그러나 툴 생성 시 반드시 명확한 설명이 포함되어야 하며, 해당 tool은 단독으로 사용되지 않고 반드시 create react agent 등과 결합하여 운용되어야 한다. agent가 쓸수 있는 도구로 변환해주는 역할이다.

셋째, create react agent는 LLM과 다양한 도구를 결합하여 통합 에이전트를 생성하는 방식이다. 이 구조는 복수의 도구를 유기적으로 활용할 수 있고, 시스템의 확장성이 뛰어나다는 장점이 있다. 다만, 초기 설정이 복잡하고 전체 구조의 난이도가 높아질 수 있으므로, 설계 단계에서 신중한 구성이 필요하다. 즉 하나의 Agent가 만들어진다.

이처럼, 각 방식은 Agentic RAG/워크플로우에서 사용될 수 있으며, 적용 위치, 장단점, 그리고 설계상 주의할 점이 상이하므로, 시스템의 규모와 목적에 따라 적합한 방법을 선택하는 것이 바람직하다.

여기서, create react agent를 LangGraph의 노드로 만들어서 실행하였을때, tool에서 틀렸던 코드를 수정하여 실행시켜 줬던점이다. 일반 llm을 명시하는 코드에서 mo ○el로 오타를 냈는데, 알아서 model로 바꿔서 코드를 실행시켜주는 경험이 있었다. agent로 만들어 주는 순간, LLM이 작성한 코드에도 관여를 한다는 것이 발견하였고, 이것은 앞으로 강력한 무기가 될것으로 예상된다.

Ⅳ. LangGraph 기반 RAG 구조 설계

LangGraph를 활용한 RAG 시스템에서는 route trigger, query expansion, query rewrite, 바리게이트 노드 등 다양한 워크플로우 모듈이 적극적으로 사용된다 [8]. 각 모듈은 질의 처리와 답변 정확도 향상에 기여하지만, 분기 조건이 명확하지 않거나 상태 관리가 미흡하면 무한루프와 같은 예기치 못한 오류가 자주 발생한다. 쉽게 예시를 들자면, query expansion이나 query rewrite 노드에서, 사용자의 질문을 받아, 더 나은 질문으로 바꿔달라고 하는 경우, 사용자의 의도와 다르게 변환되는 경우, 변환 경우에도 올바르게 답변하지 못해 조건분기가 동작하지 않아 동일한 루트를 순환하며 무한루프에 빠질 수 있다. 바리게이트 노드에서 이후 경로로의 진행을 차단하도록 설계할 수 있는데, 이 차단 이후 사용자의 입력을 반복적으로 “질문 재작성(query rewrite)” 노드로 되돌려 보내거나, 바리게이트 노드 자체로 재진입하도록 설정할 경우, 입력 수정이 없을 때 워크플로우가 ‘바리게이트 ↔ 질문 재작성’ 사이를 계속 순환하며 무한루프에 빠지게 되는 현상이 발생할 수 있다. 실제로, 사용자가 잘못된 입력을 계속 수정하지 않을 때 워크플로우는 종료되지 않고, 동일한 노드를 무한 반복하게 된다.

또 다른 사례로, 워크플로우에서 여러 외부 API 호출 노드(예: 검색 도구, 문서 요약 도구, 번역 도구 등)가 연속적으로 배치되어 있는 경우, 각 노드의 실패/재시도 로직이 명확하게 제한되지 않으면 하나의 API 호출이 실패할 때마다 재시도를 무제한으로 반복하게 되어 전체 워크플로우가 종료되지 않고 무한루프에 빠질 수 있다. API 호출 실패 시 ‘다시 시도’ 로직이 횟수 제한 없이 항상 트리거되도록 설정되어 있으면, 일정 시간 동안 동일한 API 요청이 계속 반복되어 결국 API 호출 한도 초과 또는 시스템 과부하가 발생하는 문제가 생긴다. 예를 들어, 이러한 문제를 예방하기 위해서는 각 노드의 분기 조건과 상태 관리, 재시도 한계 및 탈출 조건을 명확히 설계하는 것이 필수적이다.

Ⅴ. 결론 및 향후연구

본 논문에서는 Fine-tuning 기반 SLM과 RAG의 구조적 차이와 장단점을 비교 분석하고, RAG와 Agentic RAG 시스템 설계 시 고려해야 할 주요 이슈들을 실험적으로 제시하였다. "다양한 실험과 분석을 통해 텍스트 분할 방식, 임베딩 모델 선택, LLM 유형 및 Agentic 설계 방식 등이 RAG 시스템의 전반적인 성능에 직접적인 영향을 미치며, 실제 운용 과정에서 무한루프, 분기 조건 관리 미흡, 도구 선택의 불일치와 같은 현실적 문제점이 나타남을 확인하였다. 특히 LangGraph와 같은

워크플로우 기반 프레임워크를 활용할 때, 각 노드의 상태 관리와 재시도 조건, 도구의 역할 정의 등 세부 설계가 시스템의 안정성과 효율성에 매우 중요하다는 점이 부각되었다. 또한, API 호출 한도 초과, 입력 무한 반복 등 다양한 장애 가능성에 대응하기 위해서는, 분기 조건의 명확한 정의와 워크플로우의 탈출 조건 설계가 필수적임을 알 수 있었다.

향후 연구 및 개발 방향으로는 Agentic RAG 시스템의 자동 오류 감지 및 회복 로직 강화, 도메인 특화 워크플로우의 최적화, 다양한 LLM 및 임베딩 모델의 정량적 성능 평가, 그리고 실시간 사용자 피드백을 반영한 동적 워크플로우 개선 등이 중요하게 제시된다. 즉, RAG와 Agentic RAG 시스템의 성능은 LLM 모델 선택, 텍스트 분할 기법, 임베딩 전략, 그리고 워크플로우의 명확한 상태 관리 및 분기 조건 설계에 의해 결정된다. 이러한 후속 연구가 이루어진다면, RAG 및 Agentic RAG 시스템의 안정성과 효율성, 그리고 실질적인 활용 가능성이 더욱 높아질 것으로 기대된다.

REFERENCES

[1] H. Soudani, E. Kanoulas, and F. Hasibi, “Fine Tuning vs. Retrieval Augmented Generation for Less Popular Knowledge,” *arXiv preprint arXiv:2403.01432*, Mar. 2024. DOI: 10.48550/arXiv.2403.01432.

[2] E. J. Hu, Y. Shen, P. Wallis, Z. A. Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “LoRA: Low Rank Adaptation of Large Language Models,” *arXiv preprint arXiv:2106.09685*, Jun. 2021. DOI: 10.48550/arXiv.2106.09685.

[3] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Kuttler, M. Lewis, W. Yih, T. Rocktaschel, S. Riedel, and D. Kiela, “Retrieval Augmented Generation for Knowledge Intensive NLP Tasks,” *arXiv preprint arXiv:2005.11401*, May 2020. DOI: 10.48550/arXiv.2005.11401.

[4] A. Singh, A. Ehtesham, S. Kumar, and T. T. Khoei, “Agentic Retrieval Augmented Generation: A Survey on Agentic RAG,” *arXiv preprint arXiv:2501.09136*, Jan. 2025. DOI: 10.48550/arXiv.2501.09136.

[5] R. Qu, R. Tu, and F. Bao, “Is Semantic Chunking Worth the Computational Cost?,” *arXiv preprint arXiv:2410.13070*, Oct. 2024. DOI: 10.48550/arXiv.2410.13070.

[6] Medium. How to Optimize Chunk Size for RAG in Production? [Internet]. Available: https://pub.towardsai.net/how-to-optimize-chunk-sizes-for-rag-in-production-fae9019796b6.

[7] S. Ghosh, C. K. Reddy Evuru, S. Kumar, S. Ramaneswaran, D. Aneja, Z. Jin, R. Duraiswami, and D. Manocha, “A Closer Look at the Limitations of Instruction Tuning,” *arXiv preprint arXiv:2402.05119*, Feb. 2024. DOI: 10.48550/arXiv.2402.05119.

[8] S. Jeong, J. Baek, S. Cho, S. J. Hwang, and J. C. Park, “Adaptive RAG: Learning to Adapt Retrieval Augmented Large Language Models through Question Complexity,” *arXiv preprint arXiv:2403.14403*, Mar. 2024. DOI: 10.48550/arXiv.2403.14403.

김재호(Jae-Ho Kim)

수원대학교 컴퓨터학부 학사
수원대학교 컴퓨터학부 석사
수원대학교 컴퓨터학부 박사과정
Langchain Opentutorial Core Contributor
※관심분야 : 인공지능, LLM, RAG, AI Agent

김장영(Jang-Young Kim)

연세대학교 컴퓨터과학 공학사
Pennsylvania State Univ. 공학석사
State University of New York 공학박사
University of South Carolina 교수
수원대학교 컴퓨터학부 교수
※관심분야 : Big data, AI, Cloud computing, Networks