1. 서론 최근 몇 년 동안 대규모 언어 모델(LLMs)과 자연어 처리 기술은 대규모 데이터셋과 강력한 컴퓨팅 자원을 활용하여 인공지능 분야에 혁신을 가져왔습니다. OpenAI의 생성 사전 학습 변환기(GPT) 모델 시리즈는 가장 주목할 만한 LLM 모델 중 하나로, 2018년에 출시된 첫 번째 버전인 GPT-1은 변환기 아키텍처와 전이 학습 기술을 사용하여 자연어 처리 및 생성 작업에서 높은 성능을 보여주었습니다. 이후 GPT-2는 모델과 학습 데이터의 규모를 크게 확장하여 문맥 이해와 문장 생성 능력을 향상시켰습니다. 2019년에 출시된 GPT-3는 수십억 개의 매개변수와 대규모 데이터셋으로 학습되어 이전 모델보다 훨씬 더 복잡하고 다양한 정보를 이해하고 생성할 수 있는 능력을 보여주었습니다 [1–3].

이러한 발전은 자동 번역, 질문 응답, 문서 요약, 콘텐츠 생성 등 다양한 자연어 처리 응용 분야에 깊은 영향을 미쳤으며, 의료, 교육, 과학 등 다양한 분야에서 LLM이 복잡한 작업에 사용되고 있습니다 [4–8]. 그러나 이러한 사전 학습된 LLM은 점점 더 현실적인 텍스트를 생성할 수 있음에도 불구하고, 지식을 접근하고 정확하게 다루는 능력에는 한계가 있습니다 [9]. 또한, LLM은 의사 결정 과정을 명확히 설명할 수 없는데, 이는 블랙박스 문제로 알려져 있습니다. 따라서 결과의 정확성과 신뢰성을 확인하기 어렵고, 새로운 데이터를 업데이트하는 것도 어려운 과제입니다. 따라서 LLM은 사용자의 편의를 제공하지만, 특정 상황에서는 부적절하거나 잘못된 응답을 생성할 수 있습니다 [10–12].

최근 LLM 개발자들은 환각 문제, 업데이트 부족, 답변의 투명성 부족 등의 문제를 해결하기 위해 검색 증강 생성(RAG) 기술을 도입하고 있습니다 [13]. 이 기술은 자연어 처리 및 LLM 분야의 지식을 외부 지식 데이터베이스와 결합하여 응답의 품질과 관련성을 높입니다. RAG는 특히 학술 연구, 고객 서비스, 콘텐츠 생성 등 특정하고 최신 정보가 필요한 상황에서 유용합니다 [14–18].

RAG는 정보 검색 애플리케이션에 다음과 같은 이점을 제공합니다. 현대 데이터 환경은 방대한 양의 정보를 포함하고 있으며, 이를 효과적으로 활용하기 위해서는 빠르고 정확한 검색이 필수적입니다. 또한, 정확하고 관련성 있는 정보를 검색하려는 수요가 증가하고 있습니다. 따라서 RAG는 정보를 매개변수로 저장하는 LLM에서 더 빠른 업데이트와 맞춤형 검색을 제공할 수 있습니다. RAG는 필요한 정보를 민감한 데이터를 사용하지 않고 검색할 수 있도록 하므로, RAG는 기업 및 기관뿐만 아니라 개인을 위한 정보 검색 서비스에서 맞춤형 검색 기능을 제공하는 데 중요합니다.

본 연구에서는 검색 증강을 사용하여 개인화된 데이터베이스 시스템을 구현하였으며, 질문 응답(QA)을 통해 개인이 설정한 키워드를 문맥에 따라 태그하였습니다. 이는 정보 구조화를 날짜와 주제와 같은 카테고리로 구조화하는 것과 유사합니다. 이 검색 향상 시스템을 사용하는 개인은 키워드와 문맥 계층의 개인화된 데이터베이스를 제공받습니다. 그러나 LLM 프롬프트와 출력의 간단한 구현만으로는 이 시스템을 사용할 수 없습니다. 따라서 우리는 문맥 기반 검색 향상을 위해 RAG 과정을 적용하였고, 사용자 검색 기록을 지속적으로 업데이트하기 위해 NoSQL 데이터베이스를 구현하였습니다. 이를 통해 개인화된 데이터베이스와 QA 시스템을 구현하고, 검색 증강 생성 평가(RAGAs) 플랫폼을 통해 그 성능을 검증했습니다 [19].

본 연구는 개인 문서 내에서 키워드를 태그하고 이 정보를 사용하여 개인화된 데이터베이스에서 개인 문서를 검색하는 인터넷 웹 서비스를 중점적으로 다룹니다(예: 문서 내에서 메모 검색). 이 플랫폼에서는 로그인한 개인이 입력한 텍스트나 외부 문서에서 참조된 텍스트가 관련 키워드와 함께 개인화 데이터베이스에 저장됩니다. 따라서 개인이나 팀이 유지하고자 하는 문서는 특정 키워드로 구조화되고 태그가 지정되며, 데이터베이스에 업데이트됩니다. 이를 통해 문서의 관련성을 이해하고 해당 문서 내의 내용을 빠르게 분석함으로써 내장된 개인 정보를 상호적으로 검색할 수 있습니다.

이와 같은 목적으로, RAG는 벡터 데이터베이스와 같은 개인화된 지식의 문맥을 식별하고 개인화된 데이터베이스에서 정확한 정보를 생성하는 데 적용됩니다. 개인이 작성하고 저장한 문서를 상호 검색하여 결과를 생성함으로써, RAG를 사용한 개인화된 데이터베이스 시스템의 가능성을 입증합니다.

RAG는 지식 기반에서 다양한 정보를 검색하기 때문에 추가적인 모델 학습이나 미세 조정이 필요하지 않습니다. 따라서 본 연구에서 개발한 RAG 기반 개인화된 검색 시스템은 학습 데이터와 상관없이 축적된 정보를 통해 정확한 답변을 생성할 수 있습니다. 또한 최신 데이터를 실시간으로 반영할 수 있어 비용을 절감하고 자원을 더 효율적으로 사용할 수 있습니다. 이러한 기능은 사용자에게 더 직관적이고 효율적인 검색 경험을 제공하고, 만족도를 높이며, 필요한 정보를 쉽게 얻을 수 있도록 합니다.

이 논문의 나머지 부분은 다음과 같이 구성됩니다. 2장에서는 관련 연구를 소개하고, 3장에서는 본 연구에서 구현한 RAG의 개발을 설명합니다. 4장에서는 실제 서비스 방법을 제시하고, 원하는 결과를 제공하는 애플리케이션 설계에 LLM과 함께 사용할 수 있는 방법을 설명하며, RAGAs 프레임워크를 적용하여 성능과 응답 시간을 평가합니다. 마지막으로 5장에서는 연구를 마무리하고 향후 연구 방향을 제시합니다.

1. 관련 연구 2.1. LLM의 환각 대규모 언어 모델(LLM)은 자연어 처리 분야에서 혁신적인 발전을 가져왔지만, 실제 사실에 기반하지 않은 일관된 거짓 정보나 논리적 오류, 존재하지 않는 데이터를 생성하는 '환각' 문제를 겪고 있습니다. 예를 들어, LLM은 존재하지 않는 인물이나 사건에 대한 정보를 제공할 수 있습니다.

환각에는 사실이 아닌 정보를 포함한 '실제 환각', 논리적으로 일관성이 없거나 무의미한 답변을 포함한 '논리적 환각', 잘못된 형식의 응답을 포함한 '구조적 환각'이 있습니다. 최근 이러한 문제는 ChatGPT가 그럴듯하게 들리는 거짓 정보를 생성하는 경향 때문에 더욱 두드러졌습니다. 환각은 잘못된 정보를 퍼뜨리고 LLM이 생성한 콘텐츠의 신뢰성을 저하시킬 수 있는 심각한 영향을 미칠 수 있습니다 .

한 연구에서는 ChatGPT에 다양한 도메인의 54가지 프롬프트를 입력하여 실험한 결과, ChatGPT 3.5와 4는 각각 약 61.3%와 72%의 성공률을 보였으며, 이는 각각 39%와 28%의 환각률을 나타냅니다 . 이러한 결과는 프롬프트에 따라 다를 수 있지만, LLM의 환각률을 시사합니다.

환각의 원인은 다양하며, 주요 요인으로는 다음과 같습니다 :

* **훈련 데이터 편향**: 대부분의 LLM은 인터넷에서 수집한 대량의 텍스트 데이터를 학습합니다. 이러한 데이터에는 불완전하거나 편향된 정보가 포함될 수 있어, 모델이 잘못된 패턴을 학습할 가능성이 높아집니다.
* **모델의 추론 방식**: LLM은 다음 단어를 예측하는 방식으로 작동하므로, 주어진 문맥에서 가장 그럴듯한 단어를 생성하려고 시도하다가 문맥에 맞지 않거나 사실이 아닌 정보를 생성할 수 있습니다.
* **상호작용의 한계**: LLM은 인간 사용자와의 상호작용을 통해 정보를 생성하며, 이 정보는 제한적일 수 있어, 생성된 응답이 부정확할 수 있습니다.

따라서 프롬프트에서 정보를 추상화하여 LLM이 중요한 세부 사항을 간과하거나 잘못된 훈련 데이터로 인해 왜곡된 답변을 생성할 수 있습니다. 이러한 문제를 해결하기 위해 많은 연구가 진행되고 있습니다.

Liu 등은 LLM이 생성한 코드의 환각을 요약하고 분류하기 위해 HALLU-CODE라는 벤치마크를 사용하여 LLM 코드 생성에서 환각의 분포를 분석했습니다 . Chen 등은 LLM이 생성한 답변의 신뢰성을 감지하기 위해 ReID라는 평가 모델을 제안하고 다양한 LLM 생성 응답에서 환각을 성공적으로 감지했습니다 . Yao 등은 무작위 토큰으로 구성된 의미 없는 프롬프트가 LLM이 환각을 생성하도록 유도할 수 있음을 시연했으며, Zhang 등은 환각이 지식의 부족뿐만 아니라 모델이 초기 실수를 정당화하려고 할 때도 발생할 수 있음을 보였습니다 . Varshney 등은 '부정'이 LLM 환각에 미치는 영향을 연구하고, 부정적인 표현과 관련된 환각이 LLM의 주요 결점임을 밝혔습니다.

LLM 환각의 주요 원인은 데이터베이스나 검색 엔진과 달리 LLM이 입력된 프롬프트에서 추론을 통해 텍스트를 생성하기 때문입니다. LLM은 이미 학습한 정보에 기반하여 답변을 생성하므로 새로운 데이터를 사용할 수 없으며, 정확한 출처를 명확하게 인용할 수 있는 능력이 부족합니다. 본 연구에서는 기존 ChatGPT 모델의 환각, 업데이트 부족, 불명확한 출처 문제를 해결하기 위해 검색 증강(RAG)을 사용한 맞춤형 데이터베이스 서비스를 구현하였습니다.

2.2. RAG 연구 RAG의 전형적인 프로세스와 RAG를 사용한 LLM 구현과 사용하지 않은 구현의 차이점은 그림 1에 나와 있습니다 . 예를 들어 사용자가 대학의 외국인 학생 수에 대해 ChatGPT에 질문할 때, ChatGPT는 사전 학습된 데이터에 의존하므로 최신 데이터를 제공할 수 있는 능력이 부족합니다. 반면, RAG는 외부 데이터베이스에서 지식을 검색할 수 있으므로, 대학 데이터베이스를 참조하여 정확한 숫자를 제공할 수 있습니다. 이처럼 RAG는 ChatGPT의 검색 성능을 향상시키고, 정보 검색을 위한 맞춤형 교과서를 제공하는 것과 유사합니다.

Chen 등은 RAG의 다양한 구성 요소를 분석하고, 모듈을 새로 추가하거나 세부 조정 기술을 통합하여 RAG의 유연성을 높이는 모듈식 RAG 기술을 소개했습니다 . Zhao 등은 RAG의 근본적인 과정을 분류하고 다양한 검색 및 생성기 증강 방법론을 조사하여 실용적인 RAG 애플리케이션 및 벤치마크를 소개했습니다 . Xia 등은 도메인 정제와 RAG를 통합하여 태풍 재해에 대한 정확한 정보를 제공하는 QA 방법을 구현했습니다 . Li 등은 다양한 텍스트 생성 작업을 위한 검색 증강 텍스트 생성 및 기타 주목할 만한 접근법을 검토했습니다 .

이 연구는 RAG를 활용하여 차별화된 검색 및 QA 서비스를 제공하는 맞춤형 데이터베이스를 구축하고, 검색 정확도를 개선할 수 있음을 보여줍니다.

**3. RAG 파이프라인의 구현**

**3.1. 시스템 개요**

이 섹션에서는 그림 2에 설명된 대로 제안된 시스템에 사용된 방법을 설명합니다.

**그림 2. 시스템 아키텍처.**  
제안된 시스템은 사용자가 개인화된 텍스트 문서를 효과적으로 관리할 수 있도록 설계되었습니다. 사용자는 문서에서 특정 문장을 선택하여 관련 태그와 링크와 함께 저장할 수 있는 '태그 박스' 시스템을 통해 개인 보관용으로 저장할 수 있습니다. 이는 개인이나 팀의 특정 목적을 위해 분류된 맞춤형 지식 베이스를 나타냅니다. 이 기술은 대규모 지식 관리 엔진의 일부로, 개인이나 팀이 필요로 하는 많은 문서가 조직되고 저장되며, 특정 문장을 태그하는 데 사용되는 키워드는 메모로 작용합니다. 이 연구의 저자들은 기존 LLM의 단점을 극복하기 위해 RAG 파이프라인을 사용하여 개인화된 데이터베이스 구축 및 검색 기술을 개발하고자 했습니다.

RAG를 사용하여 LLM의 환각을 보완하고 QA 과정에서 적절하고 정확한 답변을 보장하기 위해 잘 구축된 의미적 공간이 필요합니다. 태그 서비스는 RAG를 더욱 활성화하고 성공적인 답변 생성을 위한 개인화된 의미적 공간을 구축하려고 합니다.

제안된 RAG 파이프라인은 세 가지 주요 구성 요소를 사용하여 사용자 질문을 효과적으로 처리하고 관련 답변을 생성합니다. 첫 번째는 개인화된 문서와 정보를 개인의 신원에 따라 저장하는 SQL 데이터베이스입니다. 이 SQL 데이터베이스의 사용자 정보 테이블은 '태그 박스' 시스템을 통해 등록된 사용자의 정보를 관리합니다. 사용자가 QA 시스템의 사용자 인터페이스에 질문을 입력하면 SQL 데이터베이스가 사용자 ID와 일치하는 정보를 테이블에서 추출하고 추가 처리를 위해 이를 청크로 분할합니다.

두 번째 구성 요소는 SQL 데이터베이스에서 추출한 데이터 청크를 임베딩 벡터로 변환하여 정보를 재구성하는 벡터 데이터베이스입니다. 데이터를 벡터로 저장하면 비구조화된 데이터를 처리할 수 있습니다. 벡터 데이터베이스는 벡터 데이터를 통한 유사성 검색을 사용하여 사용자의 쿼리를 처리하며, 이는 쿼리에 대한 정확한 일치보다 더 유연하게 결과를 반환할 수 있는 이점을 제공합니다.

또한, 우리는 MongoDB를 구현하여 QA의 채팅 기록을 저장하고 사용자가 나중에 사용할 수 있는 답변을 생성했습니다. LLM은 상태를 저장하지 않기 때문에 대화에서 이전 메시지를 기억하지 않습니다. 개발자는 기록을 유지하고 LLM에 대한 맥락을 제공할 책임이 있습니다. 이전의 맥락 정보를 지속 가능한 데이터베이스에 저장하고 새로운 대화에서 이를 복원하여 사용자의 질문과 답변을 요약하고 기록을 추적할 수 있는 시나리오를 허용합니다.

마지막으로 QA 생성기는 GPT3.5 터보와 같은 LLM을 사용하여 질문에 대한 정확하고 유용한 답변을 생성합니다. 이 과정은 벡터 데이터베이스에서 선택된 문맥에 따라 수행되며, 최종 답변은 사용자에게 반환됩니다.

**3.2. 데이터 추출: RAG 및 LangChain 통합**

RAG를 구현하는 간단한 방법은 외부 도구를 통합하여 환경을 만드는 강력한 프레임워크인 LangChain을 사용하는 것입니다. 이 하위 섹션에서는 RAG 구현과 LangChain을 통합한 데이터 추출 기술에 대해 자세히 설명합니다. 이 과정은 다음 단계로 구성됩니다:

* **SQL 데이터베이스에서 데이터 추출**: 이 단계에서는 사용자 정보 테이블에서 쿼리를 통해 관련 데이터를 추출합니다. 여기에는 사용자가 태그한 문서와 같은 정보가 포함됩니다.
* **청킹 및 임베딩**: 추출된 데이터는 LangChain의 통합 프레임워크를 사용하여 청크로 분할되고 벡터 데이터베이스(리트리버)로 전송되어 임베딩이 생성됩니다. 이러한 임베딩은 문서 내용을 고차원 벡터로 변환하여 정보 검색 및 매칭을 개선합니다.
* **답변 생성**: 사용자가 시스템에 프롬프트를 입력하면 저장된 벡터 데이터베이스에서 관련 문맥이 검색되어 최적의 답변을 생성하는 데 사용됩니다. LangChain은 이 과정에서 사용되는 흐름을 관리합니다.

LangChain은 LLM과 애플리케이션을 통합하는 작업을 단순화하는 소프트웨어 개발 키트(SDK)로, LLM 사용이 증가함에 따라 중요성이 커지고 있습니다. LangChain은 문서를 세분화, 결합 및 필터링할 수 있습니다. 데이터는 API를 통해 기존 SQL 데이터베이스에서 수집되어 JavaScript Object Notation(JSON) 형식으로 반환되며, 그림 3에 나와 있듯이 키-값 쌍으로 구조화됩니다. 고유 식별 번호는 특정 SQL 데이터베이스 테이블에 해당하며, 제목 및 태그 이름의 형태로 사용자 정보를 생성합니다. 개인화된 데이터베이스를 구축하는 데 사용되는 주요 정보는 개인이 문서를 분류하기 위해 설정한 키워드와 같은 태그 정보가 포함된 항목 수입니다. 우리는 이를 "태그 박스"라고 부르며, 앞서 언급했듯이 인터넷 웹 서비스로 제공됩니다. 실제로 태그 박스는 개인화 정보에 참조하는 하이퍼링크로 구현됩니다.

**그림 3. JSON 형식으로 추출된 사용자 데이터(User ID: 2190).**  
그림 4는 SQL 데이터베이스에서 개인이 생성하거나 참조한 문맥을 저장한 Context가 문서 내용의 대부분을 포함하고 있으며, 이를 기반으로 문서의 최대 길이를 1000자로 설정하고 처리하기 위해 문서를 분할합니다. JSON 기반 TextSplitter는 추출된 데이터를 직접 입력으로 받아 필요한 데이터를 선택적으로 추출 및 결합하여 JSON 형식으로 반환합니다. 분할된 문서는 OpenAI API의 text-embedding-ada-002 모델을 통해 임베딩되고 벡터 데이터베이스에 저장됩니다. 이러한 데이터는 검색 후 QA 프롬프트의 맨 앞에 가장 관련성 높은 정보를 배치하는 데 도움이 됩니다.

**그림 4. 문서 임베딩 워크플로.**

**3.3. 프롬프트 지침의 역할과 중요성**

제안된 시스템에서 사용자 프롬프트는 사용자 질문에 효과적으로 응답하는 능력에 직접적인 영향을 미치기 때문에 매우 중요합니다. 이 섹션에서는 적절한 프롬프트 제공의 중요성을 설명합니다. 그림 5는 질문을 처리하고 LangChain과 MongoDB를 사용하여 응답을 생성하는 과정을 설명합니다. 검색된 질문을 기반으로 각 질문의 문맥이 설정되고 '프롬프트' 템플릿으로 정리됩니다.

**그림 5. LangChain과 MongoDB를 사용하여 사용자 프롬프트를 처리하고 답변을 생성하는 과정.**  
프롬프트는 사용자 질문과 LLM 간의 인터페이스로 사용되며, 답변 생성을 위한 기초로 사용됩니다. 제안된 시스템은 LangChain을 통해 추출된 문서 목록을 가져와 이를 프롬프트로 포맷하고 LLM에 전달합니다. LLM이 문맥에 맞는 답변을 생성하는 능력은 제공된 프롬프트의 품질과 구조에 크게 의존합니다. 프롬프트는 답변이 정확할 뿐만 아니라 질문과 관련이 있도록 보장하는 데 중요한 역할을 합니다. 그림 6은 본 연구에서 개발된 프롬프트 템플릿의 예를 보여줍니다. 프롬프트 작성 시 다음 원칙을 엄격히 준수합니다.

* **문맥 분석**: 프롬프트에 제공된 전체 문맥을 고려하여 답변을 생성해야 합니다. 이는 답변의 정확성을 보장하고, 잘못된 정보 전달을 최소화하며, 질문의 의도와 일치하는 답변을 제공합니다.
* **정보 제한**: 문맥에서 지정되지 않은 정보는 답변에 포함되지 않아야 합니다. 이는 입력된 데이터만을 기반으로 답변을 생성하고, 시스템이 질문과 관련이 없는 테이블에 접근하지 않도록 하여 데이터 유출을 방지합니다.
* **출처 인용**: 각 답변에 대한 정보의 출처를 명시하여 사용자가 출처를 이해하거나 태그 박스에서 정보를 참조할 수 있도록 합니다.
* **불확실성 인식**: 답변을 제공할 수 없는 경우, 사용자가 지정된 상자에서 태그나 콘텐츠를 검색하여 더 정확한 결과를 얻어야 한다고 알립니다.

이렇게 프롬프트가 생성되며, 그림 6은 LLM에서 사용하기 위해 생성된 프롬프트의 예를 보여줍니다.

**그림 6. 맞춤형 프롬프트 템플릿.**  
**그림 7. QA 작업을 위한 맞춤형 프롬프트 예시.**

**3.4. 이력 관리: MongoDB**

본 연구에서 개발된 QA 시스템은 채팅 기록을 관리하기 위해 NoSQL 기반 데이터베이스인 MongoDB를 사용하여 키-값 데이터를 JSON 형식으로 저장합니다. 고정된 스키마가 없으므로 다양한 유형의 데이터를 신속하고 유연하게 처리할 수 있습니다. 시스템은 질문을 받아 MongoDB에서 최근 대화의 다섯 가지 이력을 검색하고 QA 프롬프트와 함께 LLM에 입력합니다. 그 후, 사전에 정의된 SQL 쿼리를 사용하여 데이터를 추출합니다. LLM은 대화 기록을 분석하여 문맥에 맞는 적절한 응답을 생성하고, 이를 사용자에게 전달하며 MongoDB에 저장합니다. MongoDB의 유연한 데이터 처리 기능은 사용자의 상호작용 이력을 저장하고 이를 기반으로 맞춤형 응답을 생성할 수 있게 합니다.

**4. 실험**

**그림 8. 일반적인 ChatGPT와 개인화된 RAG 기반 응답.**  
그림 8의 왼쪽은 머핀 레시피에 대한 사용자 프롬프트와 ChatGPT 3.5의 출력을 보여줍니다. 이 출력은 사용자의 성격과 의도를 반영하지 않습니다. 이는 이미 학습된 매개변수에 기반하여 생성된 것이므로 문맥을 반영하지 않으며, LLM은 프롬프트를 분석하여 단순히 답변을 생성합니다. 반면, 그림 8의 오른쪽에 표시된 바와 같이, 이 시스템에서 구현된 '태그 박스'의 개인화된 데이터베이스에서 RAG를 통해 생성된 문맥 기반 응답은 개인이 생성하고 업데이트한 머핀 레시피를 출력하며, 개인과 관련된 문맥 정보만을 표시합니다. 이 시스템이 기존 LLM과 어떻게 다른지 검증하기 위해, 우리는 RAG 성능 평가 지표를 사용하여 결과를 검증했습니다.

**4.1. RAGAs 프레임워크**

우리는 제안된 시스템의 성능을 평가하기 위해 RAG 시스템의 검색 및 생성 기능을 평가하는 데 중점을 둔 RAGAs 프레임워크를 사용했습니다. RAG 파이프라인의 각 구성 요소에 대한 평가는 답변 생성과 문서 검색 두 부분으로 나눌 수 있습니다.

**그림 9. RAGAs 평가.**  
RAGAs 프레임워크에서 평가 메트릭은 "Faithfulness"와 "Answer Relevance"입니다. "Faithfulness"는 검색된 문서와 생성된 답변 간의 관련성을 평가하며, "Answer Relevance"는 생성된 답변이 질문과의 관련성을 평가합니다. 검색 과정에서는 "Context Precision"과 "Context Recall"을 사용하여 질문에 대한 검색 문서의 정확성과 회상도를 평가합니다.

| **Metric** | **Equation** |
| --- | --- |
| Faithfulness | Faithfulness score=Number of claims in the answer that can be inferred from the given contextTotal number of claims in the answer\text{Faithfulness score} = \frac{\text{Number of claims in the answer that can be inferred from the given context}}{\text{Total number of claims in the answer}}Faithfulness score=Total number of claims in the answerNumber of claims in the answer that can be inferred from the given context​ |
| Answer Relevancy | Answer relevancy=1N∑i=1NEgi⋅Eo‖Egi‖‖Eo‖\text{Answer relevancy} = \frac{1}{N} \sum\_{i=1}^{N} \frac{E\_{g\_i} \cdot E\_o}{‖E\_{g\_i}‖ ‖E\_o‖}Answer relevancy=N1​∑i=1N​‖Egi​​‖‖Eo​‖Egi​​⋅Eo​​ |
| Context Precision | Context Precision@K=∑k=1KPrecision@k×vkTotal number of relevant items in the top K results\text{Context Precision@K} = \frac{\sum\_{k=1}^{K} \text{Precision@k} \times v\_k}{\text{Total number of relevant items in the top K results}}Context Precision@K=Total number of relevant items in the top K results∑k=1K​Precision@k×vk​​ |
| Context Recall | Context recall=GT sentences that can be attributed to contextNumber of sentences in GT\text{Context recall} = \frac{\text{GT sentences that can be attributed to context}}{\text{Number of sentences in GT}}Context recall=Number of sentences in GTGT sentences that can be attributed to context​ |

* **Faithfulness**: 생성된 답변의 신뢰성을 평가하여 주어진 문맥에 기반한 관련 정보를 제공하는지 확인합니다.
* **Answer Relevance**: 생성된 답변이 질문과 얼마나 관련 있는지를 평가합니다.
* **Context Precision**: 질문에 대해 검색된 문서가 얼마나 정확한지를 평가합니다.
* **Context Recall**: 검색된 문서가 질문에 대한 답변을 구성하는 데 필요한 정보를 포함하고 있는지를 평가합니다.

**4.2. 실험 설정**

실험에 사용된 데이터는 AIHub에서 제공하는 상식 데이터셋을 '태그 박스'의 개인화된 데이터베이스로 평가했습니다. 이 데이터셋은 100개의 "Question", "Ground\_Truth", "Context" 항목으로 구성되어 있으며, 각 항목에는 실험을 위해 생성된 질문과 실제 사실에 기반한 정답이 포함되어 있습니다. 이 문맥을 기반으로 RAGAs 프레임워크의 검색 성능을 평가했습니다. LLM 개발을 위해 NVIDIA RTX 3070 GPU와 Anaconda 환경을 사용했으며, 실험에는 Ollama 라이브러리를 사용했습니다.

| **Computing** | **Version** |
| --- | --- |
| CPU | AMD Ryzen 9 5900X 12-Core Processor |
| RAM | 32 GB |
| GPU | NVIDIA GeForce RTX 3070 |
| Anaconda Python | 3.9.19 |
| Ollama | 0.2.5 |
| RAGAs | 0.1.9 |
| Chroma DB | 0.4.23 |
| LangChain | 0.2.8 |

**4.3. 다양한 모델 조합에 대한 성능 분석**

RAG를 구현하기 위해 우리는 사용자 쿼리에서 답변 생성까지의 전체 프로세스를 검증하기 위해 Gpt3.5-Turbo를 LLM 모델로, text-embedding-ada-002를 임베딩 모델로 사용했습니다. 이 섹션에서는 RAG 시스템의 확장성을 검증하기 위해 5개의 LLM과 2개의 임베딩 모델의 다양한 조합을 사용하여 RAG 프로세스를 재구성했습니다. 특히 Gpt-3.5-Turbo, Gemma-2-9b, Llama-3-8B, Mistral-7B, Qwen2-7B LLM과 OpenAI의 "text-embedding-ada-002" 및 로컬 한국어 임베딩 모델 "snunlp/KR-SBERT-V40K-klueNLI-augSTS"를 사용했습니다. 이러한 모델들은 10개의 성능 메트릭을 사용하여 분석되었습니다.

**그림 10. Faithfulness 평가 결과.**  
**그림 11. Answer Relevance 결과.**  
**그림 12. Context Recall 결과.**  
**그림 13. Context Precision 결과.**  
**그림 14. 다양한 LLM 및 임베딩 조합에 대한 데이터 항목당 처리 시간.**

전반적으로 Ada-002 + Gpt3.5-Turbo 조합이 가장 높은 성능을 보였으며, 정확성, 답변 관련성, 문맥 정확도 등 여러 성능 지표에서 다른 모델보다 우수했습니다. 또한 KR-SBERT + Gpt3.5-Turbo 조합은 가장 짧은 처리 시간을 기록하여 유용한 대안임을 나타냈습니다. 이러한 결과는 다양한 모델 조합의 성능과 처리 시간을 비교하여 최적의 모델을 선택하는 데 중요한 정보를 제공하며, 최적의 성능과 효율성을 동시에 제공할 수 있는 모델 조합을 식별하는 데 도움이 됩니다.

1. 결론 이 연구에서는 맞춤형 데이터셋을 사용하는 LLM(대규모 언어 모델)용 정보 검색 시스템을 효율적인 RAG 접근 방식을 통해 개발했습니다. RAG는 LLM의 파라미터화된 지식과 비파라미터화된 외부 지식을 결합하여 환각 문제를 완화하고 출처를 인용하여 출력의 투명성과 사용자 신뢰를 높입니다. RAG는 관련 텍스트 자료를 색인화하여 특정 도메인에 맞게 사용자 정의할 수 있습니다.

또한, 우리는 개인화된 데이터베이스를 구축하고 데이터 프라이버시와 정확성을 최적화하면서 비용을 최소화하기 위해 RAG 접근 방식을 설계했습니다. 최신 개인 정보는 LLM을 다시 학습시키지 않고 데이터베이스에 업데이트되며, 검색 기록은 개인화된 QA 시스템을 개발하기 위해 유지됩니다.

기존 데이터베이스는 LLM 지식의 확장으로 활용할 수 있어 민감한 개인 및 직장 정보에 대해 다시 학습하거나 미세 조정할 필요가 없습니다. 데이터베이스는 실시간으로 업데이트되며, 개인이 수정하거나 삭제할 수 있는 문서를 포함하여 일관된 정보 추적과 업데이트를 보장합니다.

제안된 RAG 기반 시스템은 RAGAs 프레임워크 내에서 정확성, 응답 관련성 및 문맥 관련성 측면에서 좋은 성능을 보였습니다. 그러나 문맥 회상 점수는 다소 낮았으며, 이는 향후 연구에서 다룰 예정입니다. 따라서 실험 결과, RAG 파이프라인과 결합된 LLM 응답 시스템의 한계를 확인하였으며, 이는 문맥적 정보를 사용하는 데 어려움을 나타냅니다. 그러나 이 한계는 일반화된 환각을 사용하고 최신 정보로 모델을 업데이트하여 극복할 수 있습니다.