사물인터넷융복합논문지 Vol. 12, No. 5, pp. 81-86, 2026 https://doi.org/10.20465/KIOTS.2026.12.5.081

RAG 기반 저장장치 Trace 생성 프레임워크의 설계 및 구현

이현섭*
백석대학교 컴퓨터공학부 교수

Design and Implementation of a Trace Generation Framework for Storage Systems based on RAG

Hyun-Seob Lee*
Professor, Division of Computer Engineering, Baekseok University

요 약 대규모 언어모델(LLM) 기반 RAG(Retrieval-Augmented Generation) 시스템은 대량의 문서를 Embedding Vector로 변환하여 Vector Database에 저장하고, 질의(Query)마다 유사도 검색과 갱신 작업을 반복 수행한다. 이러한 Vector 연산이 저장장치 수준에서 유발하는 접근 패턴(Access Pattern)을 파악하는 것은 시스템 성능 향상 및 자원 최적화 연구의 핵심 전제이나, RAG와 같은 최신 AI 응용의 Trace는 공개된 사례가 드물어 관련 연구에 어려움이 있다. 기존 저장장치 연구는 주로 Random 또는 Sequential 형태의 합성(Synthetic) Trace로 성능을 평가해 왔으나, 이는 실제 AI/RAG 응용의 Vector Access 특성(유사도 기반 비정형 접근, 반복적 검색/갱신 패턴 등)을 충분히 반영하지 못한다는 한계가 있다. 본 연구에서는 RAG 시스템으로부터 실제 저장장치 접근 Trace를 생성하는 Trace Generation Framework를 설계 및 구현한다. 실제 Wikipedia 문서로 지식 데이터베이스를 구축하고 Chunk 단위로 분할하여 Embedding Vector를 생성한 후, FAISS 기반 Vector Database에 저장한다. 이후 실제 RAG Query와 유사한 검색 작업을 반복 수행하여 Vector Access Pattern을 획득하고, 각 Vector Access를 NAND Flash 저장장치의 LBA(Logical Block Address)와 연결하여 저장장치 접근 Trace를 생성한다. 본 프레임워크로 생성할 수 있는 Trace는 향후 RAG 등 AI 응용을 대상으로 한 저장장치 성능 평가 및 자원 최적화 연구에 기반 자료로 활용될 수 있을 것으로 기대된다.

주제어 : 검색 증강 생성, 주소 패턴, 패턴 분석, 데이터 분석, 주소 생성기

Abstract Retrieval-Augmented Generation (RAG) systems based on large language model (LLM) convert large volumes of documents into embedding vectors, store them in a vector database, and repeatedly perform similarity search and update operations for each query. Understanding the storage-level access patterns induced by these vector operations is essential for improving system performance and optimizing storage resources. However, traces from recent AI applications such as RAG are rarely publicly available, posing a significant obstacle to related storage system research. Prior storage research has largely relied on synthetic traces—typically random or sequential access patterns—to evaluate storage performance. However, this approach fails to adequately capture the vector access characteristics unique to real AI/RAG applications, such as similarity-based irregular access and repetitive search-and-update patterns. This study designs and implements a Trace Generation Framework that produces storage access traces derived from actual RAG operations. We construct a knowledge base from actual Wikipedia documents, split the documents into chunks, and generate embedding vectors for each chunk, which are then stored in a FAISS-based vector database. Vector access patterns are obtained by repeatedly performing search operations that emulate real RAG queries, and each vector access is mapped to a Logical Block Address (LBA) on a NAND flash-based storage device to generate a storage access trace. The traces generated by this framework are expected to serve as foundational data for future research on storage performance evaluation and resource optimization for AI applications, such as RAG.

Key Words : RAG, Address Pattern, Pattern Analysis, Data Analysis, Address Generator

This paper was supported by 2026 Baekseok University Research Fund
*교신저자 : 이현섭(hyunseob@bu.ac.kr)
접수일 2026년 08월 15일 수정일 2026년 09월 30일 심사완료일 2026년 10월 17일

www.kci.go.kr

1. 서론

최근 대규모 언어모델(Large Language Model, LLM)의 발전과 함께, 모델이 사전 학습한 지식만으로는 최신 정보나 특정 도메인 지식을 충분히 반영하지 못하는 한계를 보완하기 위한 방법으로 RAG(Retrieval-Augmented Generation) 기술이 널리 활용되고 있다. RAG 시스템은 외부 지식 문서를 Embedding Vector로 변환하여 Vector Database에 저장하고, 사용자 질의(Query)가 입력될 때마다 유사도 기반 검색을 수행하여 관련 정보를 검색한 뒤, 이를 LLM의 입력으로 활용하여 응답을 생성한다. 이러한 구조적 특성으로 인해 RAG 시스템은 기존의 단순 텍스트 생성 모델과 달리, Vector 검색 및 갱신이라는 추가적인 데이터 접근 연산을 지속적으로 수반한다. 이와 같은 Vector 연산은 최종적으로 저장장치(Storage Device) 수준의 접근 요청으로 이어지며, 이때 발생하는 접근 패턴은 기존의 일반적인 데이터베이스나 파일 시스템 워크로드와는 상이한 특성을 가질 것으로 예상된다. 예를 들어, Vector 검색은 논리적으로 인접하지 않은 데이터에 대한 비정형적 접근을 유발할 수 있으며, 질의 처리 과정에서 반복적인 검색과 갱신이 동시에 발생할 수 있다. 따라서 RAG 시스템을 위한 저장장치 설계, 캐싱 정책, 스케줄링 기법 등을 연구하기 위해서는 이러한 실제 접근 패턴을 정확히 반영한 Trace가 필수적이다. 그러나 RAG와 같은 최신 AI 응용의 저장장치 접근 Trace는 현재까지 공개된 사례가 매우 드물다. 이는 실제 서비스 환경에서 수집된 Trace가 기업의 내부 데이터로 취급되어 공개되지 않거나, RAG 시스템 자체가 비교적 최근에 등장한 기술이기 때문에 관련 연구 및 데이터셋이 충분히 축적되지 못한 데 기인한다. 이로 인해 저장장치 시스템 연구자들은 RAG 응용의 실제 특성을 반영하지 못한 채, Random 또는 Sequential 형태의 합성(Synthetic) Trace를 이용하여 성능을 평가하는 경우가 많다. 이러한 합성 Trace는 구현이 간단하고 재현이 용이하다는 장점이 있으나, 실제 AI/RAG 응용에서 나타나는 Vector Access 특성을 충분히 반영하지 못하기 때문에, 이를 기반으로 도출된 저장장치 최적화 기법이 실제 환경에서도 유효한지 검증하기 어렵다는 근본적인 한계를 가진다. 이러한 문제를 해결하기 위해서는 실제 RAG 시스템의 동작을 기반으로 한 저장장치 접근 Trace를 생성할 수 있는 프레임워크가 필요하다. 이를 위해서는 실제 문서 데이터를 기반으로 지식 데이터베이스를 구축하고, 이를 Vector 형태로 변환하여 저장한 뒤, 실제 RAG Query와 유사한 검색 작업을 반복적으로 수행하여 Vector Access Pattern을 획득하는 과정이 요구된다. 나아가 이렇게 획득한 Vector Access를 실제 저장장치의 논리적 주소인 LBA(Logical Block Address)와 연결함으로써, 저장장치 연구에 직접 활용 가능한 형태의 Trace로 변환하는 과정이 필요하다. 본 연구에서는 이러한 필요성에 기반하여 RAG 기반 저장장치 Trace 생성 프레임워크를 설계 및 구현한다. 본 프레임워크는 실제 Wikipedia 문서를 기반으로 지식 데이터베이스를 구축하고, 문서를 Chunk 단위로 분할하여 Embedding Vector를 생성한 후, 이를 FAISS 기반 Vector Database에 저장한다. 이후 실제 RAG Query와 유사한 검색 작업을 반복 수행함으로써 Vector Access Pattern을 생성하고, 각 Vector Access를 NAND Flash 기반 저장장치의 LBA와 연결하여 최종적인 저장장치 접근 Trace를 생성한다. 본 연구의 주요 기여는 다음과 같다. 첫째, 실제 RAG 시스템의 동작을 기반으로 저장장치 접근 Trace를 자동으로 생성할 수 있는 프레임워크를 제안한다. 둘째, 본 프레임워크를 통해 생성된 Trace가 기존의 합성 Trace와 달리 실제 Vector Access 특성을 반영하도록 설계 구현했다. 셋째, 생성된 Trace를 향후 RAG 등 최신 AI 응용을 위한 저장장치 성능 평가 및 자원 최적화 연구에 활용할 수 있는 기반 자료로 제공한다.

2. 배경 및 관련연구

2.1 RAG 시스템

RAG(Retrieval-Augmented Generation)[1, 2]는 대규모 언어모델(LLM)이 사전 학습 과정에서 습득한 지식만으로는 최신 정보, 특정 도메인 지식, 또는 사설 데이터를 충분히 반영하지 못한다는 한계를 보완하기 위해 제안된 기법이다. RAG 시스템은 크게 검색(Retrieval) 단계와 생성(Generation) 단계로 구성된다. 검색 단계에서는 사용자 질의(Query)를 Embedding Vector로 변환한 뒤, 사전에 구축된 지식 데이터베이스에서 질의와 유사도가 높은 문서 또는 문서 조각(Chunk)을 검색한다. 생성 단계에서는 검색된 문서를 LLM의 입력 프롬프트에 함께 제공함으로써, 모델이 검색된 정보를 참조하여 보다 정확하고 최신성이 반영된 응답을 생성하도록 유도한다. 이러한 구조는 LLM을 재학습하지 않고도 최신 지식이나 도메인 특화 지식을 효율적으로 반영할 수

www.kci.go.kr

있다는 장점을 가지며, 이로 인해 질의응답 시스템, 챗봇, 문서 요약, 코드 생성 등 다양한 응용 분야에서 활용되고 있다. 그러나 RAG 시스템은 질의가 발생할 때마다 지식 데이터베이스에 대한 검색 연산을 수반하며, 지식 데이터베이스가 지속적으로 갱신되는 환경에서는 문서 추가/삭제에 따른 Vector의 삽입 및 삭제 연산도 함께 발생한다. 즉, 기존의 단순 텍스트 생성 모델과 달리 RAG 시스템은 반복적인 데이터 접근 연산을 구조적으로 내재하고 있으며, 이는 시스템 성능 및 저장장치 자원 활용에 직접적인 영향을 미치는 요인으로 작용한다.

2.2 Vector Database

RAG 시스템에서 검색 대상이 되는 지식은 일반적으로 문서를 일정 단위(Chunk)로 분할한 후, Embedding 모델을 통해 고차원 실수 벡터(Embedding Vector)로 변환되어 저장된다[7]. 이러한 Vector 데이터를 저장하고, 질의 Vector와의 유사도를 기준으로 근접 이웃(Nearest Neighbor)을 검색하기 위해 특화된 저장 및 검색 시스템이 Vector Database이다. 전통적인 관계형 데이터베이스가 정확한 값 일치(Exact Match)를 기반으로 검색을 수행하는 것과 달리, Vector Database는 코사인 유사도(Cosine Similarity)나 유클리드 거리(Euclidean Distance) 등의 척도를 이용하여 근사 최근접 이웃 검색(Approximate Nearest Neighbor Search, ANNS)을 수행한다는 점에서 근본적인 차이가 있다. 대표적인 Vector Database 및 라이브러리로는 FAISS (Facebook AI Similarity Search), Milvus, Pinecone, Weaviate[3-5] 등이 있으며, 이 중 FAISS는 대규모 Vector 집합에 대한 고속 유사도 검색을 지원하는 오픈소스 라이브러리로서 다양한 인덱싱 기법을 제공한다. Vector Database는 대량의 고차원 Vector를 메모리 및 디스크에 효율적으로 저장하고, 질의 시마다 인덱스 구조를 탐색하여 유사도가 높은 Vector들을 반환하는 과정을 거치는데, 이 과정에서 발생하는 데이터 접근은 일반적인 정형 데이터베이스의 순차적 또는 인덱스 기반 접근과는 상이한 비정형적 접근 패턴을 나타낼 수 있다[6]. 특히 Vector의 차원 수, 인덱스 구조, 검색 시 반환되는 근접 이웃의 개수(Top-k) 등에 따라 저장장치에 발생하는 접근 범위와 빈도가 달라질 수 있어, 이러한 특성을 정량적으로 분석할 필요가 있다.

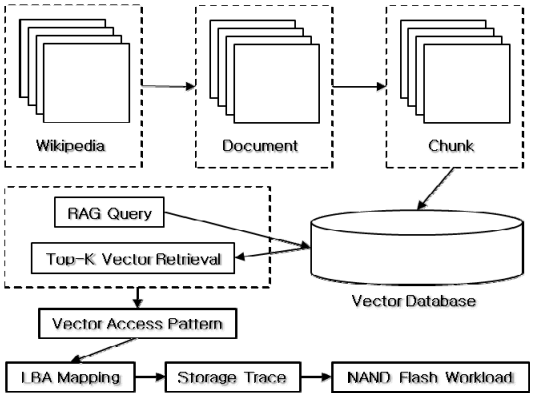
2.3 기존 저장장치 Trace 관련 연구

저장장치의 성능을 평가하고 최적화 기법을 검증하기 위해서는 실제 응용의 데이터 접근 패턴을 반영한 Trace가 필수적이다. 기존 연구에서는 이러한 Trace를 확보하는 방식에 따라 크게 실제 운영 환경에서 수집된 실측 Trace(Real-world Trace)와, 특정 통계적 분포나 규칙에 따라 인위적으로 생성된 합성 Trace(Synthetic Trace)로 구분할 수 있다. 실측 Trace는 실제 서버나 서비스 환경에서 발생하는 입출력 요청을 그대로 기록한 것으로, MSR Cambridge Trace[8, 9]나 SNIA IOTTA Trace Repository[10]에서 공개한 데이터셋이 저장장치 연구에 널리 활용되어 왔다. 또한 최근에는 Alibaba Cloud, Tencent Cloud 등 실제 클라우드 서비스에서 수집된 대규모 블록 I/O Trace를 분석하여, 부하 강도, 공간적 패턴, 시간적 패턴 등을 정량적으로 규명한 연구도 진행된 바 있다[11]. 그러나 이러한 Trace는 대부분 파일 시스템, 데이터베이스, 가상화 환경 등 전통적인 워크로드를 대상으로 수집된 것으로, 최근 부상한 AI/딥러닝 응용, 특히 RAG와 같이 Vector 연산을 수반하는 응용의 접근 특성은 반영하고 있지 않다. 또한 실제 서비스 환경에서 수집되는 Trace는 기업의 내부 데이터로 취급되어 공개가 제한되는 경우가 많아, 연구자가 최신 응용의 실측 Trace를 확보하기 어렵다는 근본적인 한계가 존재한다. 이러한 한계로 인해 다수의 저장장치 연구에서는 Random 또는 Sequential 접근 패턴을 기반으로 한 합성 Trace를 활용해 왔다. FIO와 같은 벤치마크 도구는 요청 크기, 접근 범위, Read/Write 비율 등을 사용자가 설정할 수 있도록 하여 다양한 조건의 합성 Trace를 손쉽게 생성할 수 있게 한다[14]. 이러한 방식은 구현이 간단하고 실험 조건을 정밀하게 통제할 수 있다는 장점이 있으나, 실제 응용에서 나타나는 데이터 간의 상관관계, 시간적 지역성(Temporal Locality), 공간적 지역성(Spatial Locality) 등을 충분히 반영하지 못한다는 한계를 가진다[11]. 특히 RAG 시스템에서 발생하는 Vector 검색은 유사도라는 의미적 기준에 따라 결정되므로, 단순한 확률 분포 기반의 합성 Trace로는 이러한 접근의 본질적 특성을 재현하기 어렵다.

한편, 저장장치 수준에서 발생하는 접근은 최종적으로 NAND Flash 기반 저장장치의 논리적 주소인 LBA(Logical Block Address)와 물리적 주소 간의 매핑을 관리하는 FTL(Flash Translation Layer)을 통해 처리된다[12, 13]. 최근 AI 응용의 확산과 함께 딥러닝 학습/추론 워크로드를 대상으로 한 Trace 수집 및 분석 연구가 일부 진행되고 있으나, 대부분 GPU 연산이나 모델 파라미터 접근에 초점을 맞추고 있으며, RAG 시스템의

www.kci.go.kr

핵심 구성 요소인 Vector Database의 검색 및 갱신 연산이 NAND Flash 기반 저장장치에 미치는 영향을 다룬 연구는 아직 충분히 이루어지지 않았다. 이에 따라 본 연구에서는 실제 RAG 시스템의 동작을 기반으로 Vector Access Pattern을 획득하고, 이를 저장장치 수준의 Trace로 변환하는 프레임워크를 제안함으로써, 기존 합성 Trace의 한계를 극복하고 최신 AI 응용을 반영한 저장장치 연구의 기반을 마련하고자 한다.



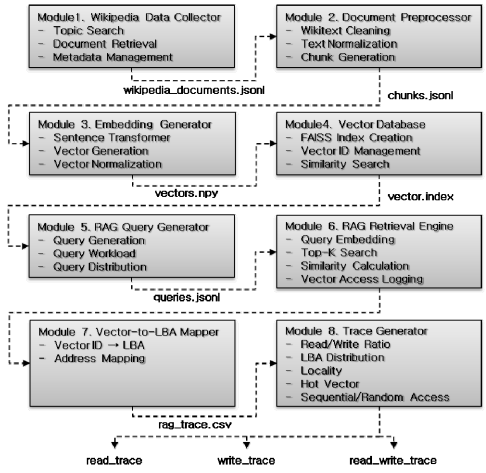
[Fig. 1] Design of RAG Trace Generator

3. RAG 기반 Trace 생성 프레임우크 설계

3.1 RAG Trace 생성기의 구조 및 설계

본 연구에서는 RAG에서 발생하는 Vector 접근 패턴을 분석하고 이를 Trace로 변환하기 위한 Trace 생성기를 설계하였다. 프로그램은 실제 Wikipedia[15] 문서를 기반으로 Vector Knowledge Base를 구축하고, RAG Query를 수행하는 과정에서 발생하는 Vector Read/Write 동작을 수집하여 저장장치 접근 Trace로 변환하도록 설계했다. FAISS 인덱스는 Flat 방식의 Inner product 인덱스를 사용했다. 임베딩 모델과 차원수는 sentence-transformers/all-MiniLM-L6-v2 모델[7] 과 384차원으로 설정했다. Top-K는 20개의 벡터가 검색되도록 하였다.

Fig. 1은 RAG Trace 생성기의 설계를 보여주고 있다. 그림과 같이 전반적인 설계는 Wikipedia로부터 문서를 수집한다. 그 다음 불필요한 Markup을 제거한 후 일정 크기의 Chunk로 분할한다. 그리고, 각 Chunk는 Embedding 모델을 이용하여 고차원의 Vector로 변환한다. 이렇게 생성된 Vector를 FAISS 기반 Vector Database에 저장한다. 이후 미리 생성된 Query Workload를 Vector Database에 입력하여 Top-K 검색을 수행하고, 검색된 Vector의 ID, 검색 순위, 유사도 및 Query ID를 기록한다.



[Fig. 2] Data Structure and Processing

3.2 Trace 생성기의 주요 모듈 및 데이터 구조

Fig. 2는 시스템 내부 주요 모듈과 산출 데이터를 보여주고 있다. 그림과 같이 내부 모듈은 Wikipedia Data Collector, Document Preprocessor, Embedding Generator, Vector Database, RAG Query Generator, RAG Retrieval Engine, Vector-to-LBA Mapper, Trace Generator 8개로 구성되어 있다. 먼저 Wikipedia Data Collector는 Wikipedia로부터 실제 RAG Knowledge Base를 구성하기 위한 원천 데이터를 수집한다. 데이터 수집 방식은 Wikipedia REST API를 이용하여 정해진 Topic(Artificial Intelligence, Machine Learning, Computer Science, SSD, NAND flash memory 등 임의의 키워드)을 검색한다.

```
{
        "document_id": 0,
        "page_id": 12345,
        "title": "Artificial intelligence",
        "source": "...",
        "content_model": "wikitext"
}
```

[Fig. 3] Example of wikipedia_documents.jsonl

www.kci.go.kr

Wikipedia로부터 검색된 데이터는 JSONL 파일로 저장된다. Fig.3는 wikipedia_documents.jsonl의 예제이다. 그림에서 필드 document_id는 내부 문서 ID를 의미한다. page_id는 Wikipedia Page ID다. title은 문서 제목이다. source는 Wikipedia 원문을 의미한다. content_model은 문서 Content Model을 의미한다. 프로그램에서 JSONL 형식을 사용하는 이유는 문서 하나를 하나의 Record로 저장할 수 있어 대규모 데이터 처리 및 부분적인 데이터 수정에 유리하기 때문이다.

```
{
        "chunk_id": 125,
        "document_id": 10,
        "page_id": 123456,
        "title": "NAND flash memory",
        "text": "NAND flash memory is..."
}
```

[Fig. 4] Example of chunks.jsonl

그 다음, Document Preprocessor 모듈에서는 검색된 원문의 Wikitext 형식에서 불필요한 Markup을 제거하여 Embedding 생성에 적합한 Text형태로 변환한다. 주요 제거 대상은 Comment, Template, Reference, HTML, Link, Heading이다. 이 과정을 통해 정제된 Clean Text를 일정 크기의 Chunk 단위로 분할한다. 이때 각 Chunk는 앞뒤 워드들이 일부 중첩 구간을 갖도록 분할한다. 그리고 각 Chunk에 독립적인 ID를 부여하여 JSONL 형식의 파일을 생성한다.

Fig. 4는 생성된 chunks.jsonl 파일의 예제를 보여주고 있다. 그림에서 chunk_id는 문서로부터 분할된 chunk의 ID를 의미한다. document_id는 문서의 ID를 의미한다. page_id는 문서의 어떤 Page인지 식별하기 위한 ID다. 그 외 제목과 내용은 title과 text 필드에 저장된다. 이렇게 분할된 각 Chunk는 Embedding Generator 모듈에서 고차원 Vector로 1 대 1 변환을 한다. 그리고 벡터는 vectors.npy에 저장된다. 이 단계에서 각 Vector를 4KB Logical 단위로 관리하면 각 단위는 4KB 크기의 LBA로 모델링할 수 있게 된다. 이렇게 변환된 Vector는 Vector Database 모듈에서 Index를 구성하여 검색 가능한 구조로 관리한다. 논문의 프로토타입에서는 FAISS를 사용하여 vectors.npy 파일에 대한 vector.index 파일을 생성하였다.

```
{
        "query_id": "Q000001",
        "query": "What is artificial intelligence?",
        "target_chunk_id": 125,
        "target_title": "Artificial intelligence"
}
```

[Fig. 5] Example of query.jsonl

Fig. 5는 생성된 queries.jsonl의 예제를 보여주고 있다. RAG Query Generator는 RAG 시스템에 입력되는 Query Workload를 생성한다. 이 과정은 사전에 LLM을 통해 Chunk로부터 실험 규모에 맞게 생성한다. 그리고 생성된 Query는 query.jsonl로 저장한다. 그 다음 RAG Retrieval Engine은 생성된 Query를 이용하여 Vector Database로부터 Top-K Vector를 검색한다. 이 과정에서 유사도를 기반으로 K개의 벡터를 검색하고 K값에 따라 Vector Read 수가 증가한다.

일반적인 RAG 연구에서는 Vector ID가 중요하지만, 본 연구에서는 Vector ID를 저장장치의 주소 공간으로 변환하는 것이 더 중요하다. Vector-to LBA Mapper는 각 Vector별 LBA 번호를 매핑하는 역할을 한다. 이때 Vector 번호를 LBA와 같은 번호로 매핑하는 것이 기본적인 방법이며, 옵션에 따라서는 번호가 일치하지 않도록 Random Mapping을 할 수 있도록 설계하였다. 마지막으로 Trace Generator는 앞에서 RAG의 시뮬레이션 과정에서 접근된 Vector를 LBA로 변환하는 관련 메타 정보를 포함한다. 그리고 이 정보를 기반으로 전체 Trace가 저장된 rag_trace.csv, Read Trace만 저장된 read_trace.csv, Write Trace만 저장된 write_trace.csv, Read/Write 혼합 Trace인 read_write_trace.csv를 생성한다.

	Timestamp	Operation	LBA	Size	Query ID	Vector ID	Rank	Similarity
2	0.00838	READ	654	4096	Q000000	654	0	0.909047723
3	0.008381	READ	656	4096	Q000000	656	1	0.75527674
~								
10997	0.208995	READ	64	4096	Q009980	64	17	0.435243964
10998	0.208996	READ	31	4096	Q009980	31	18	0.434693366
10999	0.208997	READ	486	4096	Q009980	486	19	0.432268858

[Fig. 6] Example of read/write trace

Fig. 6은 생성된 Trace의 발취된 일부 예제를 보여주고 있다. 그림의 결과와 같이 Trace는 Rag가 Timestamp별로 접근하는 목적, 접근 주소, 유사도와 랭크 등을 기록한다.

www.kci.go.kr

4. 결론

본 연구에서는 RAG 환경에서 발생하는 Vector 접근 패턴을 저장장치 Trace 형태로 생성하기 위한 RAG Trace 생성기를 설계하였다. 제안한 생성기는 Wikipedia 기반의 실제 문서를 수집하고, 문서 Chunk 생성 및 Embedding Vector 변환, Vector Database 구축, RAG Query 수행, Vector Retrieval 정보 수집 및 Vector-to-LBA 매핑의 과정을 통해 실제 RAG 환경에서 발생하는 Vector 접근 특성을 Trace로 변환하도록 구성하였다. 또한 생성된 Trace를 Read, Write 및 Read/Write 혼합 Trace로 분리하여 저장장치 관점에서 다양한 접근 패턴을 분석할 수 있도록 하였다. 제안한 Trace 생성기는 기존의 Random 또는 Sequential 기반 합성 Trace보다 실제 RAG 응용의 특성을 반영한 저장장치 Workload를 생성할 수 있도록 설계 구현했다.

향후에는 생성된 Trace를 기반으로 Vector의 접근 빈도와 의미적 유사성을 고려한 Flash-aware Vector Placement, Hot/Cold Vector 분리 저장, Vector Packing 및 LBA Mapping 기법 등 다양한 저장 기법으로 연구를 확장할 예정이다.

REFERENCES

[1] P.Lewis, E.Perez, A.Piktus, F.Petroni, V.Karpukhin, N.Goyal, H Küttler, M.Lewis, W.Yih, T.Rocktäschel, S.Riedel and D.Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Advances in Neural Information Processing Systems, Vol.33, pp.9459-9474, 2020.

[2] Y.Gao, Y.Xiong, X.Gao, K.Jia, J.Pan, Y.Bi, Y.Dai, J.Sun, M.Wang and H.Wang, "Retrieval-Augmented Generation for Large Language Models: A Survey," arXiv preprint arXiv:2312.10997, 2023.

[3] J.Johnson, M.Douze and H.Jégou, "Billion-Scale Similarity Search with GPUs," IEEE Transactions on Big Data, Vol.7, No.3, pp.535-547, 2021.

[4] Y.A.Malkov and D.A.Yashunin, "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.42, No.4, pp.824-836, 2020.

[5] J.Wang, X.Yi, R.Guo, H.Jin, P.Xu, S.Li, X.Wang, X.Guo, C.Li, and X.Xu, "Milvus: A Purpose-Built Vector Data Management System," International Conference on Management of Data, pp.2614-2627, 2021.

[6] S.J.Subramanya, Devvrit, R.Kadekodi, R.Krishnaswamy and H.V.Simhadri, "DiskANN: Fast Accurate Billion-Point Nearest Neighbor Search on a Single Node," Advances in Neural Information Processing Systems, Vol.32, pp.13748-13758, 2019.

[7] N.Reimers and I.Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, pp.3982-3992, 2019.

[8] D.Narayanan, A.Donnelly and A.Rowstron, "Write Off-Loading: Practical Power Management for Enterprise Storage," ACM Transactions on Storage, Vol.4, No.3, pp.1-23, 2008.

[9] D.Narayanan, A.Donnelly and A.Rowstron, "MSR Cambridge Traces (SNIA IOTTA Trace Set 388)," SNIA IOTTA Trace Repository, 2007. [Online]. Available: http://iotta.snia.org/traces/388

[10] SNIA, "IOTTA (I/O Traces, Tools, and Analysis) Trace Repository," Storage Networking Industry Association.

[11] J.Li, Q.Wang, P.P.C.Lee and C.Shi, "An In-Depth Comparative Analysis of Cloud Block Storage Workloads: Findings and Implications," ACM Transactions on Storage, Vol.19, No.2, pp.1-31, 2023.

[12] H.S.Lee and D.H.Lee,"An Efficient Index Buffer Management Scheme for Implementing a B-Tree on NAND Flash Memory", Data & Knowledge Engineering, Vol.69, pp.901-916, 2010

[13] H.S.Lee, S.W.Park and D.H.Lee, "RMSS: an efficient recovery management scheme on NAND flash memory based solid state disk," IEEE Transactions on Consumer Electronics, Vol.59, No.1, pp.107-112, 2013.

[14] J.Axboe, "Flexible I/O Tester (FIO)," https://github.com/axboe/fio

[15] Wikimedia Foundation, "Wikipedia," https://www.wikipedia.org.

이 현 섭(Hyun-Seob Lee) [종신회원]



- 2013년 2월 : 한양대학교 컴퓨터공학과 (공학 박사)
- 2012년 3월 ~ 2021년 2월 : 삼성전자 책임연구원
- 2021년 3월 ~ 현재 : 백석대학교 컴퓨터공학부 조교수

〈관심분야〉
인공지능, 저장시스템, 임베디드 시스템

www.kci.go.kr