벡터 DB에서 Disk기반 KV Cache를 활용한 고성능 LLM RAG 시스템

이형우[1], 김기현[1], 소정민[1], 차명훈[2], 김홍연[2], 김영재[1]
[1]서강대학교 소프트웨어융합대학, [2]한국전자통신연구원
{azwie, kion777, jso1, youkim}@sogang.ac.kr, {mhcha, kimhy}@etri.re.kr

High-Performance LLM RAG System Utilizing Disk-Based KV Cache in Vector Database

Hyungwoo Lee[1], Kihyun Kim[1], Jungmin So[1], Myung-Hoon Cha[2], Hong-Yeon Kim[2], Youngjae Kim[1]
[1]College of Computing, Sogang University, Seoul, [2]ETRI, Daejeon

요 약

LLM(Large Language Model)의 RAG(Retrieval-Augmented Generation)를 사용하는 환경에서 입력 프롬프트 길이와 모델 파라미터 수 지속적으로 증가하고 있다. 이로 인해 GPU 계산량 증가로 LLM의 TTFT(Time To First Token) 또한 증가 하고 있다. 본 논문은 TTFT를 줄이기 위해 벡터 DB와 연계한 Disk 기반 Key-Value Cache(KV Cache) 활용 방법인 RAG-DCache를 제안한다. RAG-DCache는 RAG 입력 프롬프트를 생성하는 특성을 활용하여 사전에 벡터DB와 연계하여 문서의 KV Cache를 계산하여 Disk에 저장하거나, 서비스 중 새롭게 생성되는 문서의 KV Cache를 Disk에 저장하여 LLM 추론 시 재활용한다. SQuAD 데이터셋을 사용하여 RAG-DCache의 성능을 평가한 결과, TTFT가 10%~20% 감소하였고 모델 크기와 배치 크기가 클수록 감소 효과가 더 크게 나타났다. 본 연구는 RAG를 사용하는 환경에서 Disk 기반 KV Cache를 활용하여 LLM 추론의 성능을 향상시키는 새로운 접근법을 제시하였다.

1. 서 론

Transformer [1] 기반 Large Language Model(LLM)은 강력하고 널리 쓰이는 인공지능 모델 중 하나이다. 사전 훈련된(Pre-trained) LLM은 질의응답, 번역, 요약 등 다양한 작업에서 뛰어난 성능을 보여주고 있다. 더욱 높은 성능을 위해 OpenAI GPT [2], Facebook OPT [3], Meta Llama [4]와 같은 대표적인 LLM의 파라미터 수는 수백만에서 수천억 단위로 증가하고 있다.

그러나 LLM 파라미터 수가 늘어나더라도, 사전 훈련 당시 학습하지 못한 정보들, 예를 들어 사전 훈련 이후에 생성된 정보, 기업 내부 정보 등에 대해서는 잘못된 응답을 하거나, 응답을 잘 하지 못한다. 이러한 문제들을 해결하기 위해 LLM의 추가적인 훈련이나 파라미터 변경 없이도 외부 데이터베이스나 문서를 활용하여 응답의 정확도와 신뢰성을 높이고, 확장 가능한 LLM 서비스를 제공하기 위한 방법으로 Retrieval Augmented Generation(RAG) [5]가 각광 받고 있다.

RAG는 사용자가 입력한 질의에 관련 문서 또는 문맥의 전체 또는 일부 텍스트를 추가하여, LLM 입력 프롬프트를 만드는 방식으로 기존 입력 프롬프트 대비 입력 토큰의 수가 증가한다. 그러나 RAG 사용으로 인한 입력 프롬프트 길이(N), LLM 파라미터 수와 Layer 수(L)의 증가는 Prefill 단계를 거쳐 첫번째 단어를 생성하는데 걸리는 시간(TTFT, Time To First Token)이 증가하는 문제로 이어진다. 이는 LLM의 Prefill 단계에서 GPU 계산 복잡도($O(L \cdot N^2 \cdot D)$, D=임베딩 차원)가 증가하기 때문이다.

본 논문에서는 RAG 기반 LLM 서비스 제공 시, Prefill 단계의 GPU 계산량을 줄여 TTFT를 줄일 수 있는 Disk기반 Key-Value Cache(KV Cache) [6] 활용 시스템인 RAG-DCache를 제안한다. 본 논문에서 제안하는 방식은 RAG 기반 LLM 추론 시, 입력 프롬프트 증강하는 것을 효과적으로 활용하는 것이다. 외부 데이터베이스나 문서의 텍스트를 CPU를 사용하여 KV Cache로 미리 생성하거나, 추론 과정에서 GPU로 생성된 KV Cache를 벡터 데이터베이스와 연계해 Disk에 저장한다. 이후, 다른 요청에 대해 입력 프롬프트를 증강할 때는 문서의 텍스트 대신 Disk에 저장된 KV Cache를 사용한다. 이를 통해 Prefill 단계의 GPU 계산량을 줄임으로써, TTFT를 감소시키고, 초당 질의 처리량(query/seconds)를 증가시킨다. RAG-Dcache 성능 평가를 위해 SQuAD[7] 데이터셋을 사용하여 TTFT와 초당 질의 처리량을 측정하였으며, 측정결과 TTFT는 10%~20% 감소, 처리량은 10%~20% 증가하였다. RAG-DCache 사용에 따른 응답의 정확도 차이는 없었으며, KV Cache를 Disk 저장하기 위한 용량은 텍스트 대비 증가하였다.

이 논문은 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No. 2022-0-00498, 거대 학습 모델 초고속 처리를 위한 고효율 AI 컴퓨팅SW 핵심 기술개발)

2. 배경지식

2.1 LLM 추론 시 KV Cache의 활용

Transformer 기반 LLM 추론은 '① 사용자 질의 또는 프롬프트 입력, ② 입력 프롬프트 전체를 Positional Encoding, Masked Attention, Feed Forward Network(FFN) 등을 거쳐, 첫 번째 토큰 생성(Prefill 단계), ③ 첫 번째 토큰의 Key, Value, Query와 입력 프롬프트 토큰들의 Key, Value를 사용하여 Attention, FFN 등의 과정을 거쳐 두 번째 토큰 생성(Decode 단계), ④ EOS(End of Sequence) 토큰이 나올 때 까지 ③ Decode 단계의 과정 반복'의 순으로 이루어 진다. 특히 ③ Decode 단계를 반복하면서 Attention을 수행할 때, 처리 속도를 높이기 위해 이전 까지 생성된 Key, Value 값을 GPU 메모리에 KV Cache로 저장하여 재사용한다.

2.2 RAG 사용 시 LLM prompt의 구조

RAG 시스템 구축할 때, 최신 정보나 기업 내부 정보 등 LLM에 추가로 입력할 문서들을 일정 크기로 나눈뒤, 나누어진 문서를 Embedding 모델을 사용하여 벡터로 만들어 원본 텍스트와 함께 [그림 1(a)]와 같이 벡터 DB에 저장한다. RAG를 사용하여 LLM

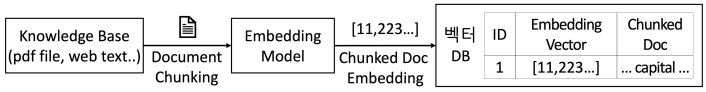
이 사용자의 질의를 처리할 때는 문서를 Embedding할 때 사용한 모델을 사용하여 사용자 질의를 벡터로 만들고, 코사인 유사도, 내적, 유클리디안 거리 등을 사용하여 사용자 질의 벡터와 가장 유사한 문서의 원본 텍스트를 추출한다. 최종적으로는 기존의 {사용자 질의 텍스트} 형태의 입력 프롬프트를 확장하여, {문서: 추출된 문서 텍스트 + 질의: 사용자 질의 텍스트 + 답변: } 형태의 입력 프롬프트가 생성된다.

3. RAG를 위한 Disk 기반 KV Cache 설계 및 구현

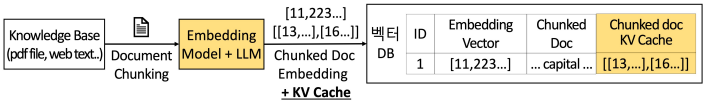
3.1 KV Cache 연계 벡터 DB의 구성

RAG에 사용되는 외부 데이터는 텍스트, 이미지 등 다양한 종류가 될 수 있다. 텍스트 데이터를 사용하여 RAG의 벡터 DB의 생성과정은 [그림 1]와 같다. LLM이 처리할 수 있는 토큰의 수는 제한이 있으므로, 외부 문서(PDF, Web문서 등)를 일정 크기로 나누고(Chunking), 나누어진 문서를 Embedding 벡터로 변환한다. 그리고 변환된 벡터, ID, 원래 문서 데이터 및 기타 메타데이터를 하나의 튜플로 저장하며, 변환된 벡터간 코사인 유사도, 유클리디안 거리, 내적 등을 활용하여 인덱스를 구성한다.

[그림 1(b)]는 본 논문에서 제안하는 벡터 DB 생성과정을 보여준다. [그림 1(a)]와 다른부분은, 일정 크기로 나누어진 문서를 추론에 사용되는 LLM을 사용하여 Chunked doc KV cache를 생성하고, 문서의 ID와 매칭 시켜 하나의 튜플로 저장한다. 벡터 DB 생성 이후 문서 데이터는 거의 변화하지 않으므로, KV cache를 저사양 GPU, CPU 등 유휴 자원을 활용하여 사전에 생성하여 Disk에 저장할 수 있으며, 추론 과정에서 GPU를 통해 생성된 KV Cache를 벡터 DB에 추가하여 Disk에 저장하는 것도 가능하다.



(a) 일반적인 벡터 DB



(b) KV Cache를 추가 활용하기 위한 벡터 DB

그림 1: LLM RAG를 위한 벡터DB 생성과정 및 튜플 구조

3.2 RAG-DCache 구성요소 및 동작

[그림 2]는 RAG를 활용한 LLM 추론 시, KV Cache 연계 벡터 DB가 포함된 RAG-DCache의 설계 및 동작방식을 보여준다. 주요 구성요소로는 KV Cache Manager, RAG Processor, KV Cache가 통합된 벡터 DB가 있다. KV Cache Manager는 문서를 KV Cache로 만들어 벡터 DB에 저장하거나, 저장된 KV Cache를 Disk로 부터 읽어 오는 기능을 한다. Memory cache는 Disk를 읽는 횟수를 줄이기 위해, 일부 KV Cache를 CPU Memory에 캐싱하는 용도로 사용된다. RAG Processor는 벡터 DB 검색 및 KV Cache 활용 LLM 입력 프롬프트 생성, Memory cache 사용율을 높이기 위한 처리 순서 조정(Dynamic batch) 등을 수행한다.

RAG-DCache의 동작 순서는 다음과 같다. ❶ 문서 등 기존 데이터를 사용하여 KV Cache Manager를 통해 KV Cache 연계 벡터 DB를 사전에 생성한다. ❷ 사용자 질의(User Query)가 입력되면, ❸ RAG Processor가 ❶ 에서 만들어진 벡터 DB에서 질의와 가장 관련있는 문서의 ID를 검색한다. ❹ 여러 사용자 질의를 처리하는 경우, Dynamic batch를 통해 같은 ID를 참조하는 질의를 함께 처리하도록 순서를 조정한다. ❺ , ❻ RAG Processor는 검색된 ID를 KV Cache Manager에 전달하고, KV Cache Manager는 ID에 해당하는 KV Cache를 Disk 또는 Memory cache에서 읽어 RAG Processor에 전달한다. ❼ RAG Processor는 사용자 질의를 LLM 모델에 맞게 embedding한 후, 사용자 질의 embedding 값과 관련된 문서의 KV Cache를 사용하여, LLM 입력 프롬프트를 생성한다. ❽ LLM 입력으로 KV Cache는 past_key_values, 사용자 질의 embedding은 Input ids 설정한 후❾ Prefill 단계와 Decode 단계를 거쳐 응답을 생성한다. 이 때 Prefill 단계에서 관련 문서의 Key-Value를 GPU에서 계산하는 대신, KV Cache를 사용함으로써 TTFT와 GPU 계산량이 줄어든다.

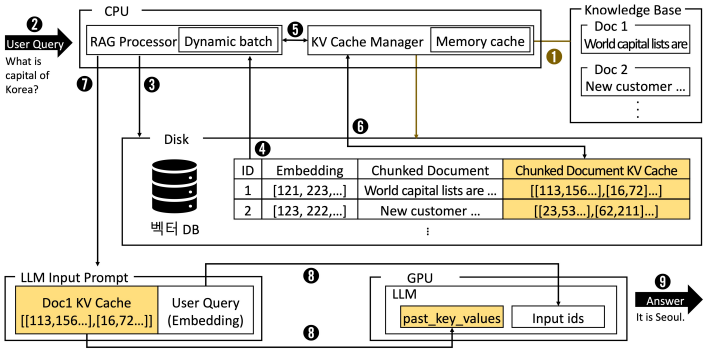


그림 2: RAG-DCache 설계 및 동작 방식

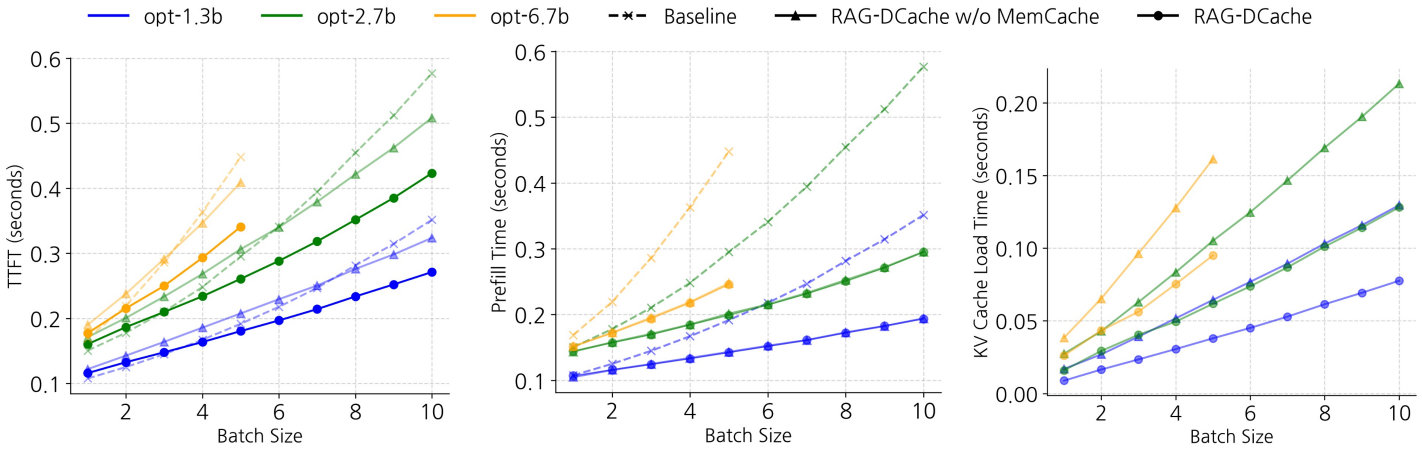
3.3 Disk에 저장된 KV Cache 적재 오버헤드 최소화

RAG-DCache의 사용은 일반적인 방식의 RAG에 대비하여 KV Cache를 GPU Memory로 읽어와야 하는 오버헤드가 생긴다. 이 오버헤드를 최소화하기 위해 RAG Processor의 Dynamic batch와 KV Cache Manager의 Memory cache를 설계하였다. Dynamic batch는 같은 문서를 사용하는 사용자 질의들을 같은 배치 또는 근처의 배치로 처리 순서를 조정함으로써 Memory cache 사용율을 높여 Disk를 읽는 오버헤드를 줄여준다. Memory cache는 Disk에서 읽어온 KV Cache를 일정 크기의 CPU 메모리에 저장하고, 이후 같은 KV Cache에 대한 요청이 있을 때 재사용한다.

4. 실험 및 평가

4.1 실험환경

실험은 8GB의 메모리를 가진 GPU 2식, 12코어 24스레드 CPU 2식, 메인메모리 64GB, Samsung 970 EVO 500GB NVMeSSD가 장착된 서버를 사용하였다. 데이터셋은 SQuAD, LLM은 Facebook의 opt 모델, 벡터 DB는 Faiss[8]를 사용하였다. 또한



(a) TTFT (b) Prefill Time (c) KV Cache Loading Time

그림 3: LLM 및 배치 크기별 TTFT, Prefill Time, KV Cache Loading Time 측정 결과

SQuAD 데이터셋에 포함된 질의와 관련된 문서를 LLM을 사용하여 사전에 KV cache로 생성하여 Disk에 저장한 후 사용하였으며, KV Cache Manager의 Memory cache는 16GB를 사용하였다. [표 1]은 실험환경에 대한 상세 사양이다.

비교 평가는 일반적인 RAG와 RAG-DCache를 사용하여 추론을 하는 경우에 대하여, LLM 크기와 배치 크기를 변화시키면서 TTFT와 초당 질의 처리량을 측정하였다.

표 1: 실험에 사용된 하드웨어(H/W) 및 소프트웨어(S/W)

H/W	**CPU**	AMD Ryzen 9 3900XT 12-Core 2식
	GPU	Nvidia GeForce RTX2080 SUPER 2식
S/W	**LLM**	facebook/opt-1.3b, 2.7b, 6.7b
	임베딩 모델	all-MiniLM-L6-v2
	데이터셋	SQuAD v1.1 Train 데이터셋 2,000개
	벡터 DB	Faiss DB, IndexFlatIP 인덱스

4.2 결과 및 분석

[그림 3]은 LLM과 배치 크기별 TTFT, Prefill Time, KV Cache Loading Time을 측정한 결과를 보여준다. 일반적인 RAG를 사용하는 것을 Baseline으로 하였다. RAG-DCache를 사용하는 경우, TTFT는 Prefill Time과 KV Cache Loading Time의 합으로 계산된다. [그림 3(a)]에서 보듯이, 일부의 경우를 제외하고 RAG-DCache를 사용하는 것이 TTFT가 감소하였다. 이는 Disk KV cache를 사용하여 Prefill Time이 줄어들고([그림 3(b)]), Memory cache 사용으로 Disk를 읽는 횟수가 줄어들어([그림 3(c)]) 전체적인 TTFT가 감소하였기 때문이다. 전반적으로 모델과 배치 크기가 클수록 RAG-DCache 사용으로 인한 Prefill Time, Disk KV Cache Loading Time의 감소 효과가 증가 하였고, 이로 인해 TTFT는 모델 크기에 따라 Baseline 대비 10%~20% 감소 하였다.

[표 2]는 RAG-DCache 사용유무에 따른 모델 크기별 평균 초당 질의 처리량을 보여준다. 처리하는 데이터셋과 배치 크기는 모델별로 모두 동일하므로, 처리량은 배치 크기에 상관없이 모델별 평균값으로 산정하였다. RAG-DCache를 사용하는 경우 질의 처리량은 10%~20% 증가하였다.

표 2: Baseline과 RAG-DCache의 평균 처리량 비교

구 분	opt-1.3b	opt-2.7b	opt-6.7b	평 균
Baseline	23.74	15.32	9.55	17.53
RAG-DCache	26.63	18.01	11.05	20.07

5. 결론 및 향후 연구

본 연구에서는 RAG를 사용하는 환경에서, 질의 관련 문서에 대한 Disk 기반 KV Cache를 벡터 DB와 연계하여 활용하는 RAG-DCache를 제안한다. KV Cache를 Disk에 저장하고, LLM 추론 시 활용함으로써 TTFT 감소와 처리량 증가 효과가 있었으며, 모델의 크기와 배치 크기가 커질수록 그 효과는 더 높아지는 것을 확인하였다. 향후에는 질의 관련 다중 문서 추출을 위한 KV Cache 생성, Disk 사용에 따른 오버헤드 최적화 방법에 대해 살펴보고, 데이터셋을 확장하여 성능 평가를 수행하고자 한다.

참고 문헌

[1] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[2] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *OpenAI Preprint*, 2018.

[3] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[4] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[5] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[6] J. Rasley, O. Ruwase, S. He, S. Ye, G. Huang, J. Liu, and Y. Wang, "Deepspeed inference: Enabling efficient inference of transformer models at unprecedented scale," *arXiv preprint arXiv:2207.00032*, 2022.

[7] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Association for Computational Linguistics, 2016.

[8] 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.