Multi-Model RAG-DCache: 다중 LLM 모델 RAG 서빙 시스템

서동휘[1], 장성[1], 김진우[1], 김홍연[2], 김영재[1†]

서강대학교 컴퓨터공학과[1]

한국전자통신연구원[2]

{xjkkm5918, jesjsjes, jinwookim, youkim}@sogang.ac.kr

hykim@etri.re.kr

Multi-Model RAG-DCache: An Efficient RAG Serving System for Multiple LLMs

Donghwi Seo[1], Seong Jang[1], Jinwoo Kim[1], Hongyeon Kim[2], Youngjae Kim[1†]

Dept. of Computer Science and Engineering, Sogang University[1]

ETRI[2]

요 약

최근 대규모 언어모델(LLM, Large Language Model)에 외부 지식을 결합하는 RAG(Retrieval-Augmented Generation) 방식이 활발히 활용되고 있다. RAG-DCache는 이러한 RAG 파이프라인에서 자주 참조되는 문서 Chunk에 대해 디스크 기반 KV Cache를 관리하여 Prefill단계의 latency를 줄이는 시스템으로 제안되었으나, 단일 모델 환경에만 적용 가능하다는 한계가 있다. 실제 서버 환경에서는 여러 모델이 동시에 서빙 되며, 각 모델이 서로 다른 SLO(Service Level Objective)를 가지는 경우가 많다. 이에 본 연구에서는 다중 모델 환경에서 RAG를 수행할 때 KV 캐시를 이용해 효율적으로 동작할 수 있도록 확장하는 구조를 제안 한다. 제안된 시스템은 모델 식별자 기반 KV Cache 생성 및 관리 구조를 갖추고 있으며, 질의의 특성에 따른 모델 라우팅(Query Routing)기능을 가진다. 이러한 확장을 통해 모델별 캐시 충돌을 방지하고, 다양 한 모델의 SLO를 고려한 차등적 서빙이 가능함을 보인다.

1. 서 론

대규모 언어모델(Large Language Models, LLMs)은 자연어 이해와 추론 능력을 바탕으로 고객 지원 챗봇, 지능형 검색 보조, 코드 생성 등 많은 곳에 사용되고 있다. 그러나 LLM은 학습 시점 이후에 생성된 최신 정보에 접근하지 못하는 근본적인 문제를 가지고 있다. 이는 대규모 데이터를 학습하는 데 막대한 시간과 비용이 소요되어, 학습 데이터를 실시간으로 갱신하기 어렵기 때문이다. 이를 해결하기 위해, 검색 증강 생성(Retrieval-Augmented Generation, RAG)방식이 고안되었다. RAG는 외부 지식 베이스에서 관련 문서를 검색하고, 이를 프롬프트에 함께 주입하여 LLM이 최신 정보를 기반으로 응답을 생성하도록 하는 방식이다. 하지만 RAG방식은 검색된 문서를 프롬프트에 추가함으로써, 입력 토큰 길이를 크게 증가시킨다. 이는 Prefill 단계의 계산 복잡도 증가로 이어져 성능 저하가 발생한다.

디스크 기반의 KV 캐시인 RAG-DCache [1]는 문서 지역성(locality)을 이용해 자주 참조되는 문서에 대한 KV 캐시를 이용해 Prefill단계의 연산 수를 줄이고, 질의 큐 대기 시간(Query Queue Wait Time)을 활용해 문서의 KV값을 계산함으로써 질의를 처리하는데 걸리는 Latency와 Throughput을 향상시켰다. 그러나 이 시스템은 KV 캐시가 본질적으로 모델 종속적이기 때문에, 동일한 모델을 서빙하는 환경에서만 사용할 수 있다는 한계를 가지고 있다. 본 연구에서는, 다양한 LLM을 동시에 지원하면서 각 사용자의 SLO를 충족시킬 수 있는 디스크 기반 MM-DCache (Multi-Model RAG-DCache)시스템을 제안한다. 이 시스템은 모델 식별자 기반 KV 캐시 생성 및 관리 구조, 모델별 디렉터리 분리를 통한 캐시 충돌 방지, 질의의 특성에 따른 모델 라우팅(Query Routing) 기능을 구현하여 이종 모델 서빙 효율을 극대화한다.

※ 본 연구는 2025년 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원(No. 2022-0-00498, 거대 학습 모델 초고속 처리를 위한 고효율 AI 컴퓨팅 SW 핵심 기술 개발) 및 SW중심대학사업(2024-0-00043)의 지원을 받아 수행되었음

2. 연구 배경 및 동기

LLM 추론은 입력 프롬프트 전체를 병렬 처리하는 Prefill 단계와 다음 토큰을 순차 생성하는 Decode 단계로 나뉜다. Prefill 단계는 입력 토큰 길이 N 에 대해 $O(N^2)$ 의 계산 복잡도를 가진다. RAG(Retrieval-Augmented Generation) 기법[2]은 외부 문서를 프롬프트에 통합하여 LLM의 정확도를 높이지만, 이로 인해 N이 증가하여 Prefill 단계의 TTFT 지연을 야기한다. Transformer 레이어 수 L, 히든 차원 D에 대해 Prefill의 복잡도는 $O(L \cdot N^2 \cdot D)$ 로 증가하므로, 동일 문서가 반복 입력될 때의 중복 계산을 회피하는 것이 필수적이다.

이러한 문제를 완화하기 위해 RAG-DCache [1]가 제안되었다. 이는 RAG 워크로드의 '문서 지역성'(동일 문서가 자주 참조됨)에 착안하여 자주 사용되는 문서의 KV 캐시를 디스크에 저장한다. Prefill 단계에서 이 캐시를 재사용하면 Self-Attention 연산은 입력 질의(Query)에 대해서만 수행되므로 $O(N^2)$ 계산 복잡도를 크게 낮출 수 있다. Shared RAG-DCache는 이를 다중 LLM 인스턴스 환경에서도 사용할 수 있도록 확장한 시스템이다. 그러나 Shared RAG-DCache는 KV 캐시가 모델의 아키텍처 및 토크나이저에 강하게 종속되어, 동일 모델 환경에서만 동작한다는 한계가 있다. 실제 서비스 환경(예: A/B 테스트)에서는 이종 모델의 동시 서빙이 필요하므로, 모델 간 캐시 충돌을 방지하는 확장된 관리 구조가 필요하다.

3. MM-DCache 설계 및 구현

3.1 Multi-Model RAG DCache의 구조

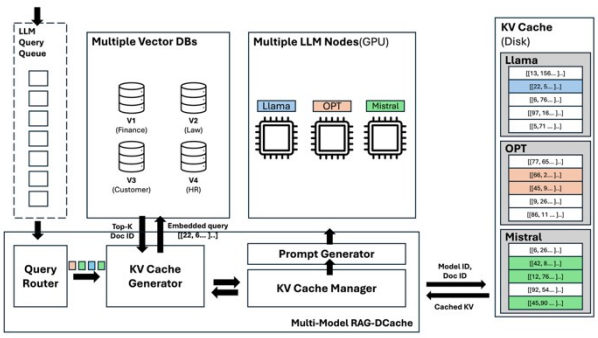

그림 1 : MM-DCache의 구조

본 연구에서 제안하는 Multi-Model RAG-DCache는 [그림 1]과 같이, Query Router와 KV Cache Generator, KV Cache Manager와 Prompt Generator의 네 가지 구성요소로 이루어져 있다. Query Router는 시스템에서 사용하는 라우팅 정책에 따라 질의를 분석하여 가장 적합한 모델로 라우팅한다. Round-Robin, Query-based, Domain-based의 세 가지 라우팅 방식을 지원하며(각 방식의 상세는 3.3 참조), 질의의 유형, 도메인, 복잡도에 따라 모델을 자동 선택한다. KV Cache Generator는 모델 식별자와 쿼리를 전달받아 Vector Embedding 및 Similarity Search를 수행하여 상위 K개의 관련 문서를 찾는다. 만약 KV캐시에 해당 문서의 KV가 존재한다면 KV Cache Manager에게 요청하고, 그렇지 않다면 새로 생성한다. KV Cache Manager는 디스크 KV 캐시에 접근하여 KV값을 저장하거나 새로 가져오는 역할을 수행한다. Prompt Generator는 검색된 문맥과 캐시된 KV 값을 결합하여 프롬프트를 구성하고 LLM Node에 전달한다.

3.2 Model-Aware KV 캐시

다중 모델 서빙 환경에서 모델 간 KV 캐시 충돌을 방지하기 위해, 기존 RAG-DCache의 캐시 관리 구조를 모델 식별자 기반 네임스페이스 구조로 확장하였다. KV Cache Manager는 생성된 KV 캐시를 model ID를 네임스페이스로 하여 모델별 전용 디렉터리에 분리 저장한다. 예를 들어, OPT 모델이 docid-001 문서를 참조할 경우, 캐시는 /cache/opt-1.3b/doc-id001.kv 경로에 저장된다. 이를 통해 서로 다른 모델이 동일한 문서 ID를 참조하더라도 캐시가 섞이지 않으며, 모델 교체나 추가 시에도 안정적인 캐시 접근을 보장한다.

3.3 Query Routing

다중 모델 환경에서는 질의의 성격에 따라 가장 적합한 모델을 선택하는 과정이 필요하다. 이를 위해 본 연구에서는 Round-robin, Query-based, Domain-based의 세 가지 간단한 라우팅 방식을 구현하였다.

Round-Robin 방식 : 가장 단순한 형태의 라우팅으로, 등록된 모델 목록을 순환하며 질의를 차례로 분배한다.

Query-based 방식 : 질의의 복잡도에 따라 모델을 선택한다. 복잡도는 질의의 의미적 복잡도와 처리상의 복잡도를 함께 고려한 개념으로, 토큰 길이와 핵심 키워드 포함 여부를 기반으로 판단한다. 간단한 질의(예: “what is”, “who is”, “when is” 등)는 단순 사실 응답에 해당하므로 소형 모델로 라우팅한다. 반면, 입력 길이가 길고 논리적 추론을 요구하는 복합 질의(“explain”, “analyze”, “compare” 등)의 경우, 더 높은 표현력과 추론 능력을 가진 대형 모델로 라우팅한다.

Domain-based 방식 : 질의가 속한 주제 영역에 따라 모델을 선택한다. 특정 도메인에 특화된 모델이 있다면, 그 도메인에 대응되는 질의 리스트를 기반으로, 입력 질의와의 코사인 유사도를 계산하여 가장 유사한 도메인의 모델로 라우팅한다. 예를 들어, 학술 응답에 특화된 모델과 대화형 모델이 있을 때 질문형, 사실기반 질의 (예:"What was the name the first cousin that Victoria married?")는 학술 응답에 특화된 모델로, '대화형' 질의 (예: "Hey, how are you doing today?" 등)는 대화형 모델로 라우팅한다.

4. 실 험

4.1 실험 환경

모든 실험은 Dual-GPU 워크스테이션 환경에서 수행되었다. 상세한 환경은 표 1과 같다.

표 1. 실험 시스템 환경

구분	사양
GPU	NVIDIA GeForce RTX 2080 SUPER (x2)
VRAM	8GB (GPU당)
CPU	*AMD Ryzen 9 3900XT 12-Core*
RAM	*64GB*
LLM Models	OPT (1.3B), DialoGPT-small
Embedding	all-MiniLM-L6-v2
Vector DB	Faiss DB
Dataset	SQuAD v1.1 Train (2,000 samples)

실험에는 총 2가지의 질문 세트를 준비하였다. 첫 번째는 질의의 난이도 기반 세트로, SQuAD v1.1 데이터셋에서 단순 사실 기반 질문 60개를 추출하고 그렇지 않은 질문 140개를 추출하여 구성하였다. 이 세트는 Round-Robin과 Query based 라우팅 정책의 평가에 사용되었다. 두 번째는 질의 도메인 기반 세트로, SQuAD v1.1 데이터셋에서 무작위로 추출한 140개의 질의(Query)와, ChatGPT를 통해 생성한 대화형 질문 60개로 구성하였다. 이 세트는 Domain-based를 평가하는데 사용되었다. 모든 실험은 동일한 조건에서 진행되었으며, TopK 값은 1로 설정하였고, 프롬프트 형식은 "Context:... Question:... Answer:"를 사용하였다. 컨텍스트 길이는 256 토큰으로 제한하였으며, 질의 간 입력 간격은 25ms로 고정하였다.

4.2 실험 결과 및 분석

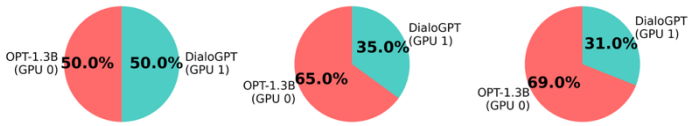

그림 2. 라우팅 정책에 따른 질의 분산 비교

(1) 라우팅 정책에 따른 질의 분산 비교 [그림 2]

Round-Robin 정책에서는 정확히 50:50의 비율로 분산되었으며, Query-based 정책의 경우 65:35를, Domain-based 정책의 경우 69:31의 분산 비율을 보였다. 입력 질의의 비율인 70:30과 비교하였을 때 꽤 높은 정확도를 보였다. Query-based의 경우 단순 패턴 매칭과 입력 토큰 수를 기준으로 분리하기 때문에 5%의 질의가 잘못 분배되었음을 확인할 수 있다. Domain-based의 경우 도메인별 질의 목록을 이용해 라우팅을 수행하여 보다 안정적인 분배 결과를 보였다.

(2) 라우팅 정책에 따른 Latency 비교 [그림 3]

각 정책 별 KV 캐시를 사용할 때와 사용하지 않을 때 평균 latency를 비교한 표는 [그림 3]와 같다. 모든 정책에서 KV

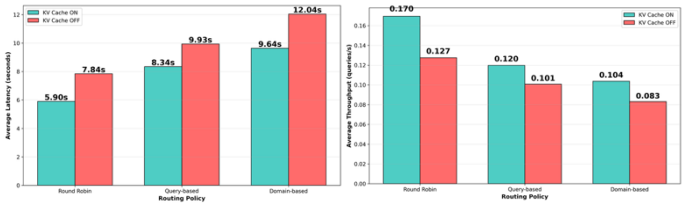

그림 3. 라우팅 정책에 따른 Latency(왼쪽)와 Throughput(오른쪽) 비교

캐시를 사용할 때 지연 시간이 유의미하게 감소하였다. Round-Robin 정책에서는 7.84초 → 5.90초(약 24.8% 감소) 로 단축되었으며,Query-based 정책은 9.93초 → 8.34초(약 16.0% 감소), Domain-based 정책은 12.04초 → 9.64초(약 19.9% 감소) 의 개선을 보였다.

(3) 라우팅 정책에 따른 Throughput 비교 [그림 3]

[그림 3]의 오른쪽이 세 정책의 Throughput을 비교한 것이다. Round-Robin 정책의 경우, 평균 처리량이 0.127 QPS → 0.170 QPS(약 33.9% 증가) 로 가장 큰 향상을 보였다. 이는 단순한 순차 분배 방식이 캐시 재활용 빈도를 극대화하여, 디스크 I/O 접근을 최소화했기 때문으로 해석된다. Query-based 정책에서는 0.101 QPS → 0.120 QPS(약 18.8% 증가), Domain-based 정책에서는 0.083 QPS → 0.104 QPS(약 25.3% 증가) 의 개선이 나타났다.

5. 결론

본 연구에서는 단일 모델 환경에 한정된 기존 Shared RAG-DCache의 한계를 개선하기 위해, 다중 모델을 동시에 서빙할 수 있는 Multi-Model RAG-DCache(MM-DCache) 구조를 제안하였다. 제안한 시스템은 모델 식별자 기반 네임스페이스를 통해 모델 간 KV 캐시 충돌을 방지하고, Query Router를 활용해 라우팅 정책별로 적절한 모델을 선택하도록 설계하였다. SQuAD v1.1 기반 실험 결과, 모든 라우팅 정책에서 질의가 안정적으로 분배되어 라우팅이 정상적으로 동작하였으며, KV 캐시 활용 시 지연 시간이 감소하고 처리량이 향상되어 제안한 구조의 효율성을 검증하였다.

참고 문헌

[1] H. Lee, K. Kim, J. Kim, J. So, M.-H. Cha, H. Kim, J. J. Kim, and Y. Kim, "Disk-based shared kv cache management for fast inference in multi-instance llm rag systems," in IEEE International Conference on Cloud Computing, 2025.

[2] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledgeintensive nlp tasks," 2021.