JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 30, No. 4: 585~590, Apr. 2026

단일 LLM 코어 공유를 통한 멀티 에이전트 시스템의 자원 사용 최적화

박현빈[1] · 김명선[2*]

Resource Optimization in Multi-Agent Systems via a Shared Single LLM Core

Hyeonbeen Park[1] · Myungsun Kim[2*]

[1]Graduate Student, Department of Applied AI, Hansung University, Seoul 02876, Korea
[2]*Associate Professor, Department of Applied AI, Hansung University, Seoul 02876, Korea

요 약

본 논문은 로컬 멀티 에이전트 시스템에서 대규모 언어 모델(LLM)을 효율적으로 운용하기 위해, 단일 LLM 코어를 운영체제의 공용 자원처럼 공유하는 중앙집중식 프레임워크를 제안한다. 에이전트별로 독립적인 LLM 인스턴스를 로딩하는 기존 방식이 GPU 메모리 중복 점유와 잦은 모델 스위칭에 따른 I/O 지연을 초래한다는 한계를 지적하고, 서버-클라이언트 아키텍처를 기반으로 단일 LLM을 GPU 상주시킨 후, FIFO 기반 스케줄링을 통해 다수 에이전트의 추론 요청을 통합 관리하도록 설계하였다. NVIDIA RTX A6000과 Llama 3.1 8B 모델을 활용한 실험 결과, 제안 방식은 에이전트 수와 무관하게 GPU 메모리 점유를 단일 모델 수준으로 유지하면서, 온디맨드 스위칭 구조 대비 최대 67.3%의 실행 지연 시간 단축을 달성하였고, 동시에 기존 개별 실행 방식과 100% 일치하는 추론 결과를 보여 정확도 저하 없이 자원 효율성과 처리 성능을 향상시킴을 확인하였다.

ABSTRACT

This paper proposes a centralized framework that shares a single large language model (LLM) core as a common system resource to efficiently operate LLM-based multi-agent systems in local environments. The conventional approach, where each agent loads an independent LLM instance, causes severe GPU memory duplication and substantial I/O latency due to frequent model switching. To overcome these limitations, the proposed server–client architecture keeps a single LLM instance persistently resident on the GPU and employs FIFO-based scheduling to centrally manage inference requests from multiple agents. Experimental evaluation using an NVIDIA RTX A6000 GPU and the Llama 3.1 8B model demonstrates that the proposed framework maintains GPU memory usage at a single-model level regardless of the number of agents and reduces execution latency by up to 67.3% compared to an on-demand switching baseline. Furthermore, it achieves inference outputs that are 100% identical to those of conventional independent execution, thereby improving resource efficiency and performance without sacrificing accuracy.

키워드 : LLM 서빙 , LLM 공유, 멀티 에이전트, 스케줄링, 시스템 자원 사용 최적화

Keywords : LLM serving, Shared LLM, Multi-agent, Scheduling, System resource optimization

Received 21 January 2026, **Revised** 3 February 2026, **Accepted** 10 March 2026

*** Corresponding Author** Myung-sun Kim (E-mail : kmsjames@hansung.ac.kr Tel +82-02 -760-4045)
Department of Applied AI, Hansung University, Seoul 02876, Korea

Open Access http://doi.org/10.6109/jkiice.2026.30.4.585 pISSN:2234-4772

Ⅰ. 서 론

최근 개인 정보 보호 강화와 저지연 서비스에 대한 필요성이 증가함에 따라, 클라우드 의존도를 낮춘 온디바이스 AI 및 로컬 환경에서의 대규모 언어 모델(LLM) 구동 기술이 주목받고 있다[1,2]. 특히 사용자의 복잡한 과업을 보조하는 멀티 에이전트 시스템의 확산에 따라, 단일 디바이스 내에서 다수의 LLM 기반 응용을 동시에 운용해야 하는 환경이 요구되고 있다.

이러한 멀티 에이전트 환경을 구축함에 있어, 에이전트마다 독립적인 LLM 인스턴스를 생성하여 운용하는 방식은 개별 에이전트의 독립성을 보장하는 데 유리하다. 그림. 1-(a)와 같이 에이전트마다 독립적으로 LLM 인스턴스를 생성하여 사용하는 방식은 특정 에이전트가 독자적인 파라미터를 사용하거나, 타 에이전트와의 간섭 없이 일관된 추론 성능을 보장받아야 하는 고성능 서버 환경에서 장점을 가진다. 그러나 자원이 제한된 환경에서는 이러한 구조가 시스템의 안정성을 저해하는 몇 가지 요인이 된다. 첫째, 비효율적인 메모리 점유가 발생한다. 동일하거나 유사한 LLM을 사용하는 에이전트들이 각각 개별적인 LLM 모델 인스턴스를 로드함에 따라 GPU 메모리 점유량이 에이전트들의 수에 비례하여 선형적으로 증가하며, 이는 빈번한 메모리 부족(Out Of Memory) 발생과 시스템 중단을 초래한다[3]. 예로서, 기존의 컨테이너 기반 가상화 환경에서는 동일 LLM 모델을 사용하는 에이전트들이라 할지라도 모델 가중치를 중복으로 로딩한다. 둘째, 입출력(I/O) 대역폭의 한계이다. 메모리 제약을 극복하기 위해 채택되는 모델 스위칭 방식은 모델 가중치를 스토리지와 GPU 메모리 사이에서 반복적으로 이동시킨다. 이 과정에서 발생하는 PCIe 대역폭의 제한과 높은 지연 시간은 LLM의 응답 성능을 심각하게 저해한다[4].

본 연구에서는 이러한 자원 제약 문제를 해결하기 위해 그림. 1-(b)와 같이 LLM을 운영체제의 공용 자원으로 관리하고 효율적으로 실행시킬 수 있는 중앙 집중식 프레임워크를 제안한다. 제안된 프레임워크는 단일 LLM 인스턴스를 GPU 메모리에 상주시킨 뒤, 다중 에이전트들의 요청을 서버-클라이언트 구조를 통해 통합 스케줄링한다. 이를 통해 에이전트 수와 무관하게 고정된 수준의 GPU 메모리 사용량을 유지하며, 모델 스위칭 오버헤드를 근본적으로 제거하여 시스템 전체의 처리 성능을 향상시킨다. 또한 LLM을 공유하더라도 개별 실행 방식과 동일 수준의 추론 정확도를 보장함을 확인하였다.

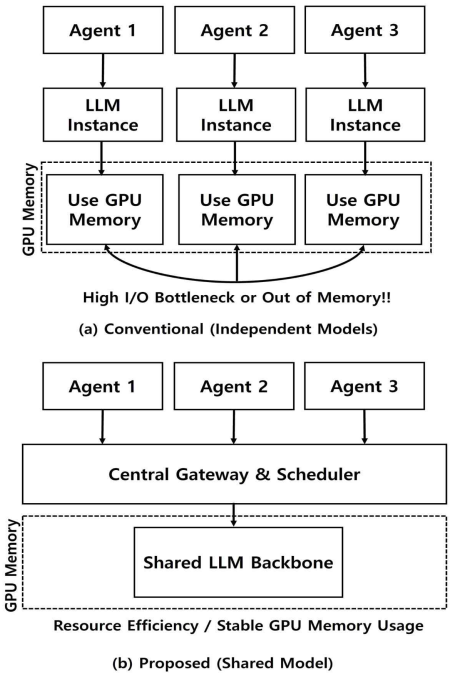

Fig. 1 Comparison of LLM serving methods

Ⅱ. 연구 배경 및 문제 정의

2.1 연구 배경

최근 LLM은 단순한 챗봇 서비스를 넘어 문서 요약, 코드 작성 보조, 실시간 번역 등 다양한 기능을 수행하는 개별 에이전트로 급격히 확산되고 있으며, 단일 기기 내에서 각기 다른 기능을 가진 복수의 에이전트를 병렬적으로 운용하려는 수요가 증대되고 있다[5,6]. 특히 최근 대다수의 특화 에이전트들이 Llama 3나 Mistral과 같은 고성능 오픈소스 모델을 백본으로 채택하여 파인튜닝하거나 프롬프트 엔지니어링을 통해 구축되는 추세를 보이고 있으며[5,7], 동일 하드웨어 자원 내에서 유사한 계열의 모델을 중복 구동해야 하는 상황이 빈번하게 발생하고 있다.

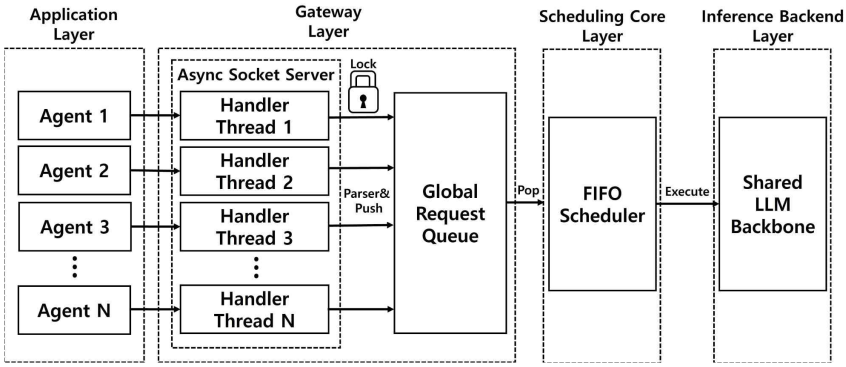

Fig. 2 Multi-agent system with a shared single LLM core

그러나 LLM 추론을 위해서는 모델의 가중치(Weights) 파라미터를 저장하는 정적 메모리와 추론 과정의 문맥 정보를 보관하는 KV 캐시 등의 동적 메모리가 GPU 내에 필수적으로 확보되어야 한다[3]. 모델의 크기가 수십 GB에 달할 정도로 대형화되면서, 개별 에이전트가 독립적인 LLM 인스턴스를 점유하려는 구조는 로컬 자원의 물리적 한계와 정면으로 충돌한다. 특히 한정된 GPU 메모리를 공유해야 하는 단일 단말 환경에서 다중 에이전트의 동시 실행은 자원 부족으로 인한 시스템 중단을 야기한다. 기존의 방식은 각 에이전트가 개별적인 프로세스나 컨테이너 단위로 구동됨에 따라, 내부적으로 동일한 백본 모델을 사용하더라도 가중치를 중복으로 로딩하는 비효율성을 가진다. 이는 유사한 기능을 수행하는 복수의 에이전트 실행시 가용 자원을 심각하게 낭비하는 결정적인 원인이 된다.

2.2 문제 정의

가용한 GPU 메모리 자원이 제한된 상황에서 대규모 모델 인스턴스를 에이전트별로 독립 구동하는 기존 방식은 두 가지 치명적인 문제를 발생시킨다.

첫째, 메모리 자원의 비효율적 중복 점유이다. 동일하거나 유사한 모델을 사용하는 다수의 에이전트가 각자 인스턴스를 로딩할 경우 중복된 모델 가중치가 메모리를 점유하게 된다. 이는 시스템이 수용할 수 있는 동시 에이전트 수를 제한할 뿐만 아니라, 가용 범위를 초과하는 순간 메모리 부족 발생 및 시스템 중단을 초래한다. 자원 증설은 비용 및 환경적 제약으로 인해 항상 실현 가능한 해결책이 될 수 없으므로, 동일 가중치의 중복 적재를 방지할 관리 체계가 필요하다.

둘째, 빈번한 모델 스위칭에 따른 데이터 전송 오버헤드이다. 메모리 부족 상황을 회피하기 위해 작업 시점에만 모델을 로딩하는 온디맨드 방식은 에이전트 교체 시마다 모델 가중치를 스토리지와 GPU 메모리 사이에서 반복적으로 이동시킨다. 이 과정에서 발생하는 데이터 전송 지연은 실제 추론 연산 시간보다 모델 로딩 시간이 지배적인 I/O Bound 상황을 유발한다. 특히, 로컬 디스크의 읽기 대역폭이 GPU 메모리 대역폭에 비해 현저히 낮으므로, 빈번한 스위칭은 하드웨어 자원의 심각한 유휴 상태(Idle)를 야기한다[4].

본 연구에서는 이러한 자원 제약 상황에서도 다중 에이전트 실행 가능성을 확보하고 모델 스위칭에 의한 데이터 전송 오버헤드를 제거하기 위해, LLM 인스턴스를 시스템 공용 자원으로 상주시킨 뒤 중앙에서 통합 스케줄링하는 프레임워크를 제안한다. 이를 통해 자원 효율성을 극대화하고 전체 추론 실행 성능을 최적화하고자 한다.

Ⅲ. 단일 LLM 코어 공유를 통한 멀티 에이전트 시스템

본 장에서는 제한된 로컬 GPU 자원 환경에서 멀티 에이전트의 효율적 운용을 위한 프레임워크를 제안한다. 단일 LLM 인스턴스를 다수의 에이전트가 공유하게 함으로써 자원 효율성을 극대화하고 모델 중복 로딩 및 스위칭에 따른 병목 현상을 해결한다.

3.1 계층별 프레임워크 설계

그림. 2는 제안하는 프레임워크의 계층 구조를 나타내며, 이는 모듈성과 확장성을 고려해 네 가지 계층으로 구분하여 설계되었다.

첫째, 애플리케이션 계층은 최상위 계층으로, 사용자 인터페이스와 개별 에이전트들이 구동되는 환경이다. 에이전트는 독립적인 LLM 모델 가중치를 가지지 않고, 필요한 추론 요청을 텍스트 프롬프트 형태로 구성하여 소켓 통신을 통해 서버로 전송한다.

둘째, 게이트웨이 계층은 외부 클라이언트의 요청을 시스템 내부로 연결한다. 비동기 소켓 서버를 통해 다수 에이전트의 동시 접속을 수용한다. Handler Thread는 서버에 연결된 에이전트와 1:1로 대응하여 생성되며 서버로 유입된 프롬프트 데이터를 파싱하여 표준화된 쿼리 객체(Query Object)로 캡슐화한 후 전역 요청 대기열(Global Request Queue)로 전달한다.

셋째, 스케줄링 코어 계층은 공유 자원에 대한 접근권을 통제하는 중앙 제어부로, 전역 요청 대기열과 FIFO 스케줄러를 통해 작동한다. 먼저 전역 요청 대기열은 스레드 안전(Thread-safe)한 구조를 갖춰 다수의 클라이언트로부터 동시다발적으로 유입되는 쿼리의 충돌을 방지하기 위해 버퍼링 작업을 수행하고 입력 순서를 유지한다. 이후 FIFO 스케줄러는 대기열의 쿼리를 순서대로 인출하여 처리하되, 한 번 실행이 시작되면 해당 추론이 완전히 종료될 때까지 GPU 자원을 독점적으로 할당하는 방식을 통해 작업의 순차적 처리를 보장한다.

넷째, 마지막 추론 백엔드 계층은 실제 연산을 수행하는 물리적 실행 환경이다. 시스템 부팅 시점에 공유하는 LLM 모델을 GPU 메모리에 단 한 번 적재하고 상주시킨다. 이후 모든 스케줄링 작업은 GPU 메모리에 존재하는 모델을 공유하여 연산을 수행하게 되며, 기존 방식의 핵심 문제였던 모델 로딩 및 스위칭 오버헤드를 근본적으로 제거한다.

3.2 FIFO 기반 스케줄링

일반적인 시분할(Time-slicing) 방식의 스케줄링을 적용한다면 다중 사용자의 응답성을 높일 수 있겠으나, LLM 추론 환경에서는 문맥 전환(Context Switch) 시마다 KV 캐시의 백업 및 복구라는 추가적인 오버헤드를 유발한다. 자원이 제한된 로컬 환경에서 이러한 전환 비용은 전체 시스템의 처리량을 떨어뜨리는 주요 원인이 된다.

본 프레임워크는 제안하는 LLM 공유의 효율성을 극대화하고 하드웨어 자원을 오직 추론 연산에만 집중시키기 위해 FIFO(First-In-First-Out) 정책을 채택한다. FIFO 정책은 하나의 요청을 중단 없이 빠르게 완결함으로써 전체적인 작업 소요 시간을 최적화할 뿐만 아니라, 에이전트 간 데이터 격리 측면에서도 이점을 가진다. 특정 에이전트의 추론이 수행되는 동안 GPU 자원을 독점적으로 할당하고, 작업 완료 직후 해당 세션의 KV 캐시 및 중간 연산 데이터를 메모리상에서 명시적으로 초기화(Flush)함으로써 에이전트 간 컨텍스트 혼입이나 데이터 노출 위험을 구조적으로 차단한다. 그 결과, 문맥 전환에 따른 불필요한 데이터 이동을 배제함과 동시에, 공유 환경에서도 각 에이전트의 독립적인 실행 성능과 데이터 무결성을 최적화하였다.

Ⅳ. 실험

4.1 실험 환경 설정

본 연구에서 제안하는 프레임워크의 성능 및 자원 효율성을 검증하기 위해 단일 서버 환경에서 실험을 진행하였다. 하드웨어 환경은 48GB의 메모리를 탑재한 NVIDIA RTX A6000 GPU 1개를 사용하였으며[8], 소프트웨어 환경은 Ubuntu 22.04 LTS 운영체제 기반에서 Python 3.11.13, PyTorch 2.1.0, CUDA 11.8 버전을 사용하여 구축하였다. 실험 모델은 Meta의 Llama 3.1 8B [7]를 선정하였으며, 가중치 변조에 따른 변수를 배제하기 위해 FP32 정밀도를 기준으로 모든 추론을 수행하였다. 비교 대상이 되는 기존 방식은 에이전트별로 독립된 프로세스 환경을 할당하여 개별 런타임을 가지는 구조를 채택하였으며, 메모리 부족 상황을 해소하기 위해 모델 가중치를 시스템 메모리와 GPU 메모리 사이에서 주기적으로 교체하는 온디맨드 스위칭 방식을 적용하였다. 또한 프롬프트의 길이, 내용에 따라 추론 시간이 달라지므로, 모든 실험 시나리오에서 사용하는 입력 프롬프트는 완전히 동일하게 구성하였다.

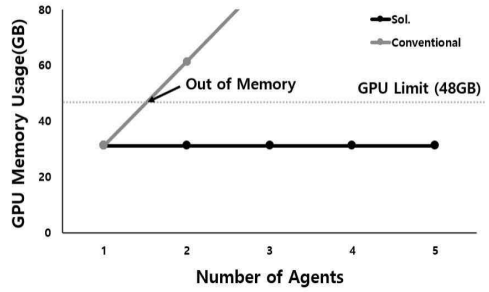

Fig. 3 GPU memory usage versus number of agents

4.2 에이전트 개수에 따른 GPU 메모리 사용량

논문의 핵심 기여점인 메모리 효율성을 확인하기 위해 동시 운용할 에이전트의 수를 1개에서 최대 5개까지 증가시키며 GPU 메모리 요구량을 측정하였다. 실험 결과, 기존의 방식에서는 에이전트의 수가 증가함에 따라 GPU 메모리 요구량이 약 30.67GB의 정수 배로 선형 증가하며 2개 이상의 에이전트 구동 시 모델 적재 중 메모리 초과가 발생하였다. 반면 제안된 시스템(Sol.)은 다수의 애플리케이션이 실행되더라도 시스템 초기 부팅 시 점유된 모델 메모리를 안정적으로 유지하였다. 이는 모델 백본을 공유함으로써 운용중인 에이전트 수와 무관하게 고정된 수준의 자원 점유가 가능함을 입증하고, 로컬 환경에서 멀티 에이전트 확장성을 확보할 수 있음을 보여준다.

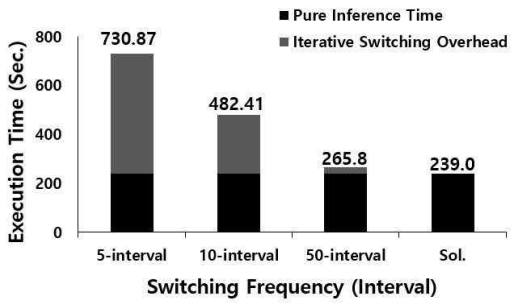

Fig. 4 Execution time versus model switching frequency

4.3 모델 스위칭에 의한 지연시간 분석

본 연구가 지향하는 제한된 자원 환경에서는 다중 LLM의 동시 적재가 불가능하여 Out-Of-Memory 오류가 발생하므로, 시분할 스위칭 방식이 유효한 대조군 중 하나이다. 따라서 본 실험은 스위칭이 강제되는 상황을 베이스라인으로 설정하여, 제안 기법이 필연적으로 발생하는 스위칭 오버헤드를 얼마나 효과적으로 제거하는지를 입증하는 데 목적이 있다. 이를 검증하기 위해 2개의 에이전트가 각각 5, 10, 50회 간격으로 모델을 교체하는 다양한 혼잡도 시나리오에서 성능을 비교하였다. 실험 결과, 제안 시스템은 스위칭 빈도가 가장 높은 5-interval 시나리오 대비 67.3%의 실행 시간을 단축하였다. 추가적으로, 본 실험에서 측정된 스위칭 오버헤드는 1회당 약 25.8초이며 PyTorch의 to() 메서드를 통한 개별 텐서 단위의 메모리 복사 과정과, empty_cache() 호출에 따른 GPU 캐싱 할당자의 물리적 메모리 재할당 지연이 결합된 결과로 분석되었다.

Table. 1 Comparison and validation of inference accuracy across execution scenarios

Methods	Token-level Match
Sol.	Reference
5-interval	Identical to reference
10-interval	Identical to reference
50-interval	Identical to reference

4.4 추론 정확도 검증

모델 공유 방식이 기존의 독립 실행 방식과 동일한 추론 결과를 도출하는지 검증하기 위해 정확도 비교 실험을 수행하였다. 추론의 결정론적 특성을 보장하기 위해 매 시점 가장 확률이 높은 토큰만을 선택하는 방식(Greedy Decoding)을 적용하였으며, 두 방식의 출력값이 텍스트 수준에서 완전히 일치하는지 대조하였다. 실험 결과, 표. 1에 나타난 바와 같이 제안된 시스템의 결과는 개별 실행 방식의 출력과 100% 일치하였다. 이는 본 시스템이 채택한 FIFO 스케줄링 기반의 정책에 기인한다. 해당 정책은 특정 에이전트의 추론이 완료될 때까지 GPU 자원을 독점적으로 할당함으로써, 다중 에이전트 간의 작업 실행 시점을 시간적으로 완벽히 분리한다. 결과적으로 제안된 프레임워크의 구조적 격리는 컨텍스트 충돌을 원천적으로 방지하며, 추가적인 관리 계층을 경유함에도 불구하고 모델 본연의 추론 능력을 유지함을 입증한다. 다만, 이러한 격리 방식은 추론의 신뢰성을 확보하는 대신 자원 활용의 유연성 측면에서 일부 한계점과 단점을 가진다. 선행

작업의 처리 시간이 길어질 경우 후행 작업의 대기 시간이 급격히 증가하는 Head-of-Line Blocking 현상이 발생할 수 있으며, 이는 긴 추론 태스크가 자원을 독점할 때 시스템 전체의 공정성(Fairness)을 저해할 수 있는 요인이 될 수 있다.

Ⅴ. 결 론

본 연구는 로컬 환경의 자원 제약을 극복하기 위해 단일 LLM 인스턴스를 상주시킨 뒤 공유하는 중앙집중식 프레임워크를 제안하였다. 실험 결과, 본 시스템은 구동중인 에이전트 수와 무관하게 GPU 메모리 점유를 단일 모델 수준으로 고정하여 자원 효율성을 극대화하였다. 성능 면에서 모델 스위칭 오버헤드를 근본적으로 제거해 빈번한 모델 전환이 이루어지는 기존 방식보다 최대 67.3%의 실행 지연 시간을 단축하였다. 또한, LLM 공유 구조 도입 시에도 원본 모델과 100% 일치하는 정확도를 유지함을 입증하였다. 결과적으로 제안된 LLM 공유 프레임워크는 정확도의 손실 없이 속도와 메모리 효율을 동시에 충족한다.

ACKNOWLEDGEMENTS

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS-2025-23963679).

REFERENCES

[1] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, “Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing,” *IEEE*, vol. 107, no. 8, pp. 1738-1762, Aug. 2019. DOI: 10.1109/JPROC.2019. 2918951.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637-646, Oct. 2016. DOI: 10.1109/JIOT.2016.2579198.

[3] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, “Efficient Memory Management for Large Language Model Serving with PagedAttention,” in *Proceedings of the 29th Symposium on Operating Systems Principles*, Koblenz: DE, pp. 611-626, 2023. DOI: 10.1145/3600006.3613165.

[4] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, “Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey,” *IEEE*, vol. 108, no. 4, pp. 485-532, Apr. 2020. DOI: 10.1109/JPROC.2020. 2976475.

[5] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, “Large Language Models: A Survey,” *arXiv preprint arXiv: 2402.06196*, Feb. 2024. DOI: 10.48550/arXiv.2402.06196.

[6] J. Kaddour, J. Harris, M. Mozes, H. Bradley, R. Raileanu, and R. McHardy, “Challenges and applications of large language models,” *arXiv preprint arXiv: 2307.10169*, Jul. 2023. DOI: 10.48550/arXiv.2307.10169.

[7] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, et al., “The Llama 3 Herd of Models,” *arXiv preprint arXiv: 2407.21783*, Jul. 2024. DOI: 10.48550/arXiv.2407.21783.

[8] NVIDIA. NVIDIA RTX A6000 Datasheet Report. Retrieved from https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/quadro-product-literature/proviz-print-nvidia-rtx-a6000-datasheet-us-nvidia-1454980-r9-web%20(1).pdf.

박현빈(Hyeon-been Park)

2020년 - 2026년 2월 한성대학교 IT융합공학부 졸업
2026년 3월 - 현재 한성대학교 AI응용학과 석사과정
※ 관심분야 : 딥러닝 최적화, Linux kernel, 병렬 컴퓨팅

김명선(Myung-Sun Kim)

2000년 - 2002년 LG전자 전송연구소 주임연구원
2002년 - 2011년 삼성전자 DMC 연구소 책임연구원
2016년 서울대학교 전기컴퓨터공학부 박사 졸업.
2016년 - 2019년 삼성전자 SR연구소 수석연구원
2019 - 현재 한성대학교 AI응용학과 부교수
※ 관심분야 : NPU(인공지능 프로세서), Linux kernel, HW/SW Co-design, 객체인식, 딥러닝 가속 및 병렬 컴퓨팅