ProgressiveServe: 서버리스 LLM 콜드 스타트 완화를 위한 점진적 모델 로딩 및 복구 기법

박나담[O*], 이나경[*], 이주원[*], 심재형
이화여자대학교 컴퓨터공학과
parknd@ewhain.net, rinarina0429@ewha.ac.kr, juwonlee.cse@gmail.com, jh.sim@ewha.ac.kr

ProgressiveServe: Progressive Model Loading and Recovery for Mitigating Cold Start in Serverless LLM Serving

Nadam Park[O*], Nakyeong Lee[*], Juwon Lee[*], Jaehyeong Sim
Department of Computer Science and Engineering, Ewha Womans University

요 약

서버리스 환경에서 LLM(Large Language Model) 서빙은 유휴 상태에서도 GPU 비용을 지불하는 온프레미스 방식과 달리 실시간 사용량 기반 과금 모델을 통해 운영 비용을 줄일 수 있다는 이점이 있으나, 심각한 콜드 스타트 문제에 직면하고 있다. 컨테이너 초기화와 대용량 모델 파라미터 로딩으로 인해 첫 응답까지 수십 초 이상의 지연이 발생하며, 이는 실시간 대화형 AI 서비스를 사용하는 사용자 경험을 심각하게 훼손한다. 이를 해결하기 위해 기존 서버리스 LLM 연구는 체크포인트 병렬 로딩, 자원 스케줄링 등 시스템 경로 최적화에 집중했지만, 모델 파라미터의 대규모 전송과 적재로 인한 근본적인 지연 문제를 해소하지 못한다는 한계를 가진다.

본 논문은 콜드 스타트 지연을 완화하기 위해 프루닝된 경량 모델을 우선 로드·서빙한 뒤, 백그라운드에서 전체 모델을 점진적으로 복구하는 파이프라인 ProgressiveServe를 제안한다. ProgressiveServe의 핵심 아이디어는 단일 추론 세션 내에서 동적으로 레이어 구조를 변화시켜, 빠른 초기 응답과 높은 최종 정확도를 모두 달성하는 것이다. 먼저, 각도 기반 연속 레이어 프루닝을 통해 경량화된 모델을 우선 서빙하여 첫 토큰 응답 시간(TTFT)을 단축시킨다. 이후 사용자에게 응답을 전달하는 동안 백그라운드에서 나머지 레이어를 로딩한 후 프루닝된 모델에 결합하여 점진적으로 원본 모델로 복구시킨다. 이 과정에서 프루닝으로 인해 발생한 성능 저하는 각 레이어에 LoRA 어댑터를 부착하여 효과적으로 완화시킨다. NVIDIA RTX 5090 서버에서 Llama2-7B 모델을 이용한 실험 결과, ProgressiveServe는 QA 태스크 다운스트림 성능 저하는 최소화하면서 TTFT를 선행 연구 대비 21.1% 감소시킨다.

1. 서 론

기존의 대규모 언어 모델(LLM, Large Language Model) 서빙은 주로 충분한 자원이 확보된 온-클러스터 환경에서 이루어져 왔다. 그러나 LLM 활용이 급증함에 따라, 유휴 시간에는 비용을 절감하고 필요 시에만 자원을 할당받는 비용 탄력성에 대한 요구가 커지며 서버리스 환경에서의 LLM 연구가 주목받고 있다.

서버리스 환경에서 LLM을 활용할 때 가장 큰 기술적 난제 중 하나는 콜드 스타트(Cold Start) 문제이다. 콜드 스타트란 오랫동안 호출되지 않았던 서버리스 함수가 새로운 요청을 받았을 때, 즉시 응답할 수 있는 실행 인스턴스가 존재하지 않아 새롭게 인스턴스를 생성해야 하는 상황을 의미한다. 이 과정에서는 컨테이너 초기화뿐 아니라, 거대한 LLM 파라미터를 원격 저장소로부터 로컬 메모리로 불러와야 하므로 전체 추론 절차를 시작하기까지 상당한 지연이 발생한다.

이러한 지연은 실시간 상호작용이 핵심인 대화형 AI 서비스의 사용자 경험을 심각하게 저해하는 주요 원인이 된다. 특히 콜드 스타트 지연 시간의 대부분은 모델 파라미터를 원격 저장소에서

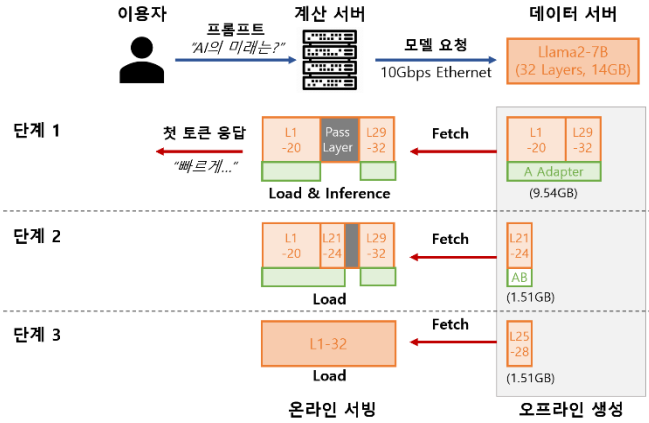

그림 1. ProgressiveServe 파이프라인 개요도

로컬 컴퓨팅 노드로 전송하는 Fetch 단계와, 이를 메모리에 적재하는 Load 단계에서 발생한다. 이는 LLM이 수십 GB에 이르는 많은 모델 파라미터를 요구하기 때문이다.

이러한 콜드 스타트 지연 문제를 해결하기 위해 기존 연구들은 체크포인트 로딩, 자원 스케줄링 등 시스템 수준의 최적화 기법에 주로 집중해 왔다 [1]. 그러나 이러한 접근은 모델 파라미터의

[*] 이 저자들은 본 연구에 동등하게 기여하였음.

대규모 전송과 적재로 인한 근본적인 지연 문제를 해소하지 못한다는 한계를 가진다.

본 연구는 이러한 한계를 극복하기 위해 LLM의 점진적(Progressive) 로드 및 복구 기법인 ProgressiveServe를 새롭게 제안한다. 제안된 기법은 대규모 원본 모델을 한 번에 로드하지 않고, 초기 응답 생성에 필수적인 일부 레이어를 제외하고 프루닝한 경량 모델을 우선 로드하여 빠른 응답을 제공한다. 이후 백그라운드에서 단계적으로 나머지 레이어를 불러와 전체 모델을 복구함으로써, 점진적으로 원본 모델의 성능을 회복하는 동적 파이프라인을 구현하였다. 이를 통해 콜드 스타트로 인한 초기 지연 시간을 크게 단축시키면서도, 최종적으로는 원본 모델과 동등한 수준의 품질을 유지할 수 있다.

2. 본 론

ProgressiveServe는 크게 오프라인 단계와 온라인 단계로 구성된다. 오프라인 단계에서는 Llama2-7B 모델을 기반으로 레이어 단위 프루닝과 LoRA(Low-Rank Adaptation) [2]를 적용하여 서빙에 필요한 구성요소들을 미리 생성·저장한다. 온라인 단계에서는 이를 점진적으로 로딩한 후 결합하여 사용자에게 완성도 높은 응답을 제공한다.

2.1. 계층적 프루닝 전략

ProgressiveServe의 핵심은 모델 사이즈를 경량화하여 콜드스타트 지연을 축소시키는 것이다. 이를 위해 본 연구에서는 LLM의 레이어별 중요도를 정량적으로 평가하는 각도 기반 연속 레이어 프루닝 방법을 적용하여 모델 프루닝을 진행한다. 이 방법은 선행 연구 [3]에서 제안된 레이어 간 표현 유사도 기반 레이어 프루닝 수식을 도입한 것이다. 구체적으로 각 레이어의 입력과 출력 간의 코사인 유사도를 계산하여, 유사도가 높은 레이어를 프루닝 대상으로 식별한 후, 해당 레이어를 제거한다. 해당 방법은 QA(Question-Answering) 성능 저하가 미미하기 때문에 모델 경량화로 인한 성능 손실을 최소화한다.

본 연구에서는 Llama2-7B [4]의 32개 레이어 중 21번부터 28번 레이어를 프루닝하여 레이어 그룹 A(1-20번, 29-32번 레이어)를 만들고, 나머지 레이어 중 21-24번 레이어를 레이어 그룹 B, 25-28번 레이어를 레이어 그룹 C로 저장한다. 이후 레이어 그룹 A를 우선 서빙하여(단계 1) 콜드 스타트 지연을 최소화하고, 이후 백그라운드에서 로딩한 레이어 그룹 B(단계 2), C(단계 3)를 순차적으로 로드하여 원본 모델을 복구한다.

2.2. LoRA 어댑터

레이어 프루닝은 모델 크기를 효과적으로 줄이지만, 필연적으로 정보 손실을 야기하여 초기 서빙되는 레이어 그룹 A의 성능을 저하시킨다. ProgressiveServe는 이를 완화하고 모델 복구 과정에서 표현력을 점진적으로 회복하기 위해, 각 복구 단계에 특화된 별도의 LoRA 어댑터를 학습하고 적용하는 계층적 성능 복구 전략을 사용한다. 구체적으로, 다음과 같이 두 개의 어댑터를 학습하고 적용한다:

1) A 어댑터: 단계 1에서 레이어 그룹 A의 성능 복구를 목표로 한다. 학습 시, 원본 모델에서 레이어 그룹 B, C를 PassLayer로 치환한 상태에서 레이어 그룹 A에만 A LoRA 어댑터를 부착하여 학습한다.
2) AB 어댑터: 단계 2에서 레이어 그룹 A, B 모델의 최적화를 목표로 한다. 학습 시, 레이어 C만 PassLayer로 치환한 상태에서 레이어 그룹 A, B에 AB LoRA 어댑터를 부착하여 학습한다.

각 어댑터는 대표적인 QA 데이터셋인 SQuAD [5]를 사용하여 PEFT(Parameter-Efficient Fine-Tuning)의 대표적인 기법인 LoRA로 학습된다. 이를 통해 프루닝으로 인한 정보 손실을 각 단계에서 효과적으로 보완하여, 초기 응답부터 원본 모델이 복구되기 전까지 전 과정에 거쳐 안정적인 성능을 보장하는 역할을 수행한다.

2.3. 점진적 레이어 복구 메커니즘

ProgressiveServe의 레이어 복구는 서비스 중단 없이 모델 품질을 향상시킨다. 본 시스템은 3단계 복구 전략을 통해 콜드 스타트 지연을 최소화하면서도 최종 단계에서는 완전한 품질의 서비스를 제공한다.

단계 1에서는 원본 모델의 설정(Config)과 레이어 그룹 A와 A 어댑터를 Fetch하고, 해당 Config를 바탕으로 모델 아키텍처 정보를 불러와 플레이스홀더(Placeholder)를 생성한다. 이후 레이어 그룹 A의 Safetensors 샤드를 순차적으로 Fetch 및 Load하여 해당 레이어 그룹의 파라미터를 채우고, 빈 레이어들의 자리에는 PassLayer를 설치한다. 마지막으로 A 어댑터를 적용하여 초기 응답 모델의 품질을 향상시킨다.

단계 2에서는 레이어 그룹 B와 AB 어댑터를 Fetch 및 Load하여 모델 성능을 단계적으로 향상시킨다. 단계 1에서 PassLayer로 채워져 있던 레이어 그룹 B 부분을 실제 레이어로 교체하며, AB 어댑터를 추가 및 활성화한다. 이때 기존 A 어댑터는 비활성화되며, AB 어댑터가 레이어 그룹 A, B 전체에 대해 최적화된 파라미터를 제공한다.

마지막 단계 3에서는 레이어 그룹 C를 Fetch 및 Load하여 PassLayer로 남아있던 모든 부분을 실제 레이어로 복구한다. 이를 통해 모델이 원본 모델과 동일한 아키텍처를 갖추게 된다. 이 시점에서 기존에 활성화되어 있던 AB 어댑터를 비활성화한다.

3. 실 험

3.1. 실험 환경

본 연구는 콜드 스타트 지연을 측정하기 위해 단일 노드에서 RayServe [6]로 GPU 1개를 점유한 Actor를 매 트라이얼마다 새로 생성하고, 모델의 최초 로딩 시간을 측정했다. 모든 실험은 Ubuntu 24.04, NVIDIA 드라이버 580.65.06, CUDA 13.0 환경에서 NVIDIA RTX 5090 GPU를 사용했고, 모델 가중치는 NFSv4로 export된 원격 데이터 서버 경로를 노드에 마운트해 FP16 정밀도의 Safetensors를 읽는다. 원격 데이터 서버와 로컬

서버간은 16Gbps 이더넷으로 연결되어 있다. 이때 마운트 옵션은 고정하여 네트워크 I/O 조건이 트라이얼 간 일관되게 유지되도록 했으며, 파일 접근 지연에 네트워크 특성이 반영되도록 하였다. 비교 선행 연구인 ServerlessLLM은 스토리지 계층으로 별도 프로세스인 sllm-store을 사용하며, 제안된 연구에서는 sllm-store 없이 동일 NFS 경로의 파일을 직접 읽도록 설정했다. 각 트라이얼 시작 시 OS 페이지 캐시와 GPU 캐시를 초기화하고, 토크나이저 로딩 또한 Hub 다운로드를 차단해 마운트된 NFS 경로에서만 참조하도록 통제하였다.

3.2. 구현

비교 대상인 ServerlessLLM은 NFS 상의 원본 체크포인트를 sllm-store를 통해 일괄 적재한다. 본 논문에서 제안한 방법인 ProgressiveServe에서는 원본 모델을 사전 프루닝, 재학습한 산출물들을 동일 NFS 경로에서 직접 읽는다. 콜드 스타트 지표는 Actor 생성 직후부터 모델 Fetch를 시작해 첫 번째 토큰의□응답이 발생하는 시점까지의 벽시계 시간(Wall-Clock Time)으로 정의한다. 각 트라이얼마다 캐시를 초기화 하였다.

단계별 복원의 정확도 영향을 검증하기 위해 TriviaQA [7] Validation 샘플을 제로샷으로 EM, F1 스코어를 평가했다. 원본 Llama2-7B를 기준으로 단계 1, 2, 3을 비교했다. 모든 평가는 동일 프롬프트와 생성 파라미터(max_new_tokens=10, greedy decoding)를 적용했다.

3.3. 실험 결과

표 1. QA 태스크 성능 및 서빙 시간 평가 결과

	단계	EM (%)	F1 (%)	벽시계 시간(s)	TTFT (s)
[1]		55.67	66.11	114	114
This Work	1	48.22	54.26	90	90
	2	56.22	62.02	19	
	3	55.67	66.11	19	

표 1은 선행 연구 ServerlessLLM과 제안된 방법인 ProgressiveServe의 실험 결과를 나타낸다. ServerlessLLM은 단일 단계로 전체 모델을 로드하여 EM 55.67, F1 Score 66.11의 성능을 보인 반면, TTFT(Time-To-First-Token)는 전체 벽시계 시간과 동일한 114초를 기록하였다. 반면 ProgressiveServe는 3단계 로딩 방식을 적용하여 단계 1에서 EM 48.22, F1 Score 54.26을 달성하고 90초 만에 첫 토큰을□생성하였다. 이는 ServerlessLLM 대비 약 21.1%의 TTFT 단축이다. 이후 단계 2와 3에서 추가 레이어를 점진적으로 로드하면서 EM은 56.22, 55.67, F1 Score은 62.02, 66.11으로 각각 개선되었다. ProgressiveServe의 총 벽시계 시간은 ServerlessLLM보다 14초 증가하였으나 이는 콜드 스타트 환경에서 빠른 초기 응답을 제공하기 위한 트레이드오프로 해석되며 사용자가 이미 응답을 받고 있기에 체감하기 어려울 것으로 예상된다. 최종 단계의 성능(EM, F1 Score)은 ServerlessLLM과 동일한 수준으로 ProgressiveServe가 모델 품질을 유지하면서도 사용자 체감 지연 시간을 효과적으로 개선할 수 있음을 보여준다.

4. 결 론

본 논문에서는 서버리스 환경에서 LLM을 서빙할 시 발생하는 콜드 스타트 지연 문제를 해결하기 위한 ProgressiveServe라는 새로운 파이프라인을 설계 및 구현하고, 성능을 검증하였다. 전체 모델을 일괄 로딩하는 기존의 연구와 달리, ProgressiveServe는 프루닝된 경량 모델을 우선 서빙하는 방식으로 대체하여 콜드 스타트 지연을 축소시켰다. 특히, 각도 기반 연속 레이어 프루닝 기법을 적용하여 모델의 깊은 레이어를 선별적으로 제거하고, 동시에 PassLayer 메커니즘을 통해 제거된 레이어 자리를 플레이스 홀더로 대체함으로써 서비스 중단 없이 점진적으로 모델 복구가 가능한 구조를 확보할 수 있었다. 전체적으로 3단계 복구 전략을 통해 Llama2-7B 모델에서 초기 응답 시간을 선행 연구 대비 21.1% 단축하면서도 최종단계에서는 원본 모델과 동일한 성능을 달성할 수 있는 기술을 개발함으로써 서버리스 LLM 서빙의 실용성을 확보하였다.

본 연구에서 개발한 점진적 레이어 복구 기술은 실시간 대화형 AI 서비스의 사용자 경험을 향상시키고, 다양한 환경에서 LLM 도입 장벽을 완화하기 위한 서버리스 기반의 고속, 저비용, 신뢰성 LLM 서빙 시스템을 구현하는 데 필수적인 핵심 원천 기술이 될 것으로 기대된다.

참 고 문 헌

[1] Y. Fu et al., “ServerlessLLM: Low-latency server-less inference for large language models,” in USENIX OSDI, 2024.
[2] E. J. Hu et al., “LoRA: Low-rank adaptation of large language models,” arXiv preprint arXiv:2106.09685, 2021.
[3] A. Gromov et al., “The unreasonable ineffectiveness of the deeper layers,” arXiv preprint arXiv:2403.17887, 2024.
[4] H. Touvron et al., “Llama 2: Open Foundation and Fine-Tuned Chat Models,” arXiv preprint arXiv:2307.09288, 2023.
[5] P. Rajpurkar et al., “SQuAD: 100,000+ Questions for Machine Comprehension of Text,” in EMNLP, 2016.
[6] P. Moritz et al., “Ray: A distributed framework for emerging AI applications,” in USENIX OSDI, 2018.
[7] M. Josh et al., “TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension,” in ACL, 2017.