모바일 환경에서 Speculative Decoding을 위한 효율적 트리 구조

전영민[0], 신동군

성균관대학교 전자전기컴퓨터공학과

mkjsym@g.skku.edu, dongkun@skku.edu

Efficient Tree Structure of Speculative Decoding on Mobile Devices

Youngmin Jeon[0], Dongkun Shin

Department of Electrical and Computer Engineering, Sungkyunkwan University

요 약

본 논문은 모바일 기기와 같이 연산 자원이 한정된 환경에서 대규모 언어 모델(LLM)의 추론 속도를 향상시키기 위한 트리 기반 추측 디코딩(Tree-based Speculative Decoding) 알고리즘의 효율적인 트리 구조를 제안한다. 제안하는 트리 구조는 불필요한 계산을 줄이고 검증 성공률을 높여 전체적인 추론 지연 시간(Latency)을 크게 단축할 수 있다. 실험 결과, 제안하는 트리 구조가 기존의 Baseline 트리 구조에 비해 모바일 환경에서 초당 토큰 생성 수(Token per Second)를 약 2.13배 향상시키면서도 모델의 생성 품질은 저하되지 않음을 확인했다.

1. 서 론[1]

최근 대규모 언어 모델(Large Language Model, LLM)은 자연어 해 및 생성 분야에서 전례 없는 성능을 보이며, 다양한 서비스의 핵심 기술로 자리 잡았다. 그러나 LLM의 자기 회귀(Auto-regressive)적 추론 방식은 전체 추론시간을 지연시키는 병목을 유발한다.

이를 해결하기 위해 여러 토큰 후보를 병렬적으로 검증하는 추측 디코딩(Speculative Decoding) 기법[1][2]이 활발히 연구되고 있으며, 특히 다수의 후보 시퀀스를 트리(Tree) 형태로 동시에 탐색하는 트리 기반 추측 디코딩은 단일 시퀀스 대비 더 많은 후보 시퀀스를 생성하여 평균 수락 길이를 극대화하고, 더 큰 속도 향상 잠재력을 가진다.

이러한 추측 디코딩 기법은 자원이 제한된 모바일 환경에서도 LLM 추론 속도를 가속하는 데 여전히 유효한 접근법이다. 하지만 기존 연구들은 풍부한 서버 자원을 가정하여 복잡한 트리 구조를 제안하는 데 초점을 맞추고 있어, 이는 자원이 극도로 제한된 모바일 환경에서는 오히려 성능 저하를 유발할 수 있다. 따라서 본 논문은 모바일 환경에서 후보 생성 비용과 검증 성공률의 최적 균형점을 찾기 위한 모바일 특화 정적 트리 구조를 제안한다.

* 이 논문은 삼성전자 미래기술육성센터의 지원을 받아 수행된 연구임(과제번호 SRFC-IT2402-15)

2. 서 론

2-1. LLM과 자기 회귀적 추론

LLM은 트랜스포머(Transformer) 아키텍처를 기반으로 방대한 양의 텍스트 데이터에 대해 사전 학습된 모델이다. LLM은 주어진 문맥(Context)을 바탕으로 다음에 등장할 가장 확률이 높은 단어, 즉 토큰(Token)을 예측한다. 대부분의 LLM은 텍스트 생성을 위해 자기 회귀 방식을 사용한다. 자기 회귀적 추론 방식은 매 스텝마다 토큰을 순차적으로 생성해야 하므로, 연산 강도(Arithmetic Intensity)가 극히 낮은 GEMV(General Matrix-Vector multiplication) 연산에 의존하게 된다. 이는 GPU의 병렬 연산 능력을 활용하지 못하고, 주로 메모리 대역폭(Bandwidth)에 의해 지연시간이 결정되는 병목 현상을 유발한다.

2-2. 추측 디코딩 (Speculative Decoding)

추측 디코딩은 이러한 자기 회귀 방식의 지연 문제를 해결하기 위해 제안된 기법으로, LLM의 최종 출력 품질을 그대로 유지하면서 추론 속도를 높이는 최적화 기법이다. 핵심 원리는 작고 빠른 드래프트 모델을 사용해 여러 토큰으로 구성된 후보

시퀀스를 미리 생성하고, 크고 정확한 타겟 모델이 이 후보 시퀀스 전체를 한 번의 GEMM(General Matrix-Matrix multiplication) 연산으로 검증하는 것이다. 이 과정에서 드래프트 모델의 예측과 타겟 모델의 예측이 일치하는 토큰들은 '수락(Accept)'되어, 여러 번의 순차적 연산을 한 번의 병렬 연산으로 대체함으로써 지연시간(Latency)을 크게 줄인다. 또한 최종 토큰 결정은 항상 타겟 모델이 하므로, 속도 향상과 함께 기존 모델의 생성 품질이 100% 보장된다는 장점이 있다.

2-3. 트리 기반 추측 디코딩

추측 디코딩은 토큰 후보를 정적(Static) 또는 동적(Dynamic) 트리 구조로 생성하여 검증 효율을 높일 수 있다. 정적 트리는 구조가 고정되어 오버헤드가 적지만, 문맥 대응력은 떨어진다. 반면 동적 트리는 문맥에 따라 실시간으로 구조를 변경하여 토큰 수락률이 높다는 장점이 있지만, 이 과정에서 발생하는 계산 오버헤드가 크다.

연산 자원이 극도로 제한된 모바일 환경에서는 동적 트리의 오버헤드가 오히려 전체 추론 시간을 증가시키는 원인이 될 수 있다. 따라서 본 논문은 구조가 단순하고 오버헤드가 적은 정적 트리가 모바일 환경에 더 적합하다고 판단하여 정적 트리 내에서 수락 길이와 드래프트 예산의 상충 관계(Tradeoff)를 고려하여 검증 지연 시간을 최적화하는 방안에 집중한다.

3. 본 론

3-1. 연산 자원

모바일 환경은 서버 대비 연산 성능(FLOPS)과 메모리 대역폭이 현저히 낮아 드래프트 예산(Budget)과 검증 지연시간 사이의 Trade-off 관계가 매우 민감하다. 그림 1이 이를 명확히 보여준다. 서버(주황색 선)는 드래프트 예산이 증가해도 지연시간이 완만하지만, 모바일(녹색 선)은 예산 8을 넘어 16부터 지연시간이 급등하고 32 이상에선 지수적으로(exponentially) 폭등한다. 이 병목의 핵심은 모바일 GPU의 제한된 온 칩 캐시 용량과 낮은 메모리 대역폭 때문이며, 실험에 사용한 Qualcomm Adreno 830 GPU의 12MB 온 칩 메모리(GMEM) 크기로 설명할 수 있다.

드래프트 예산 16의 경우 연산에 필요한 중간 활성화 값(약 0.34MB)과 새 KV 캐시(약 8.0MB)의 총합은 약 8.34MB이다. 이 수치는 12MB 캐시 용량 대부분을 차지하여, 기존 KV 캐시 읽기 및 기타 오버헤드를 처리할 여유 공간이 부족한 '캐시 압박(Cache Pressure)' 상태를 유발한다. 이것이 그림 1에서 예산 8에서 16으로 증가할 때 지연시간이 처음으로 가파르게 상승하는 이유이다. 또한 드래프트 예산 24의 경우 같은 방식으로 계산한 총 필요 용량은 약 12.51MB(활성화 0.51MB + KV 캐시 12.0MB)이다. 이 값은 12MB의 온 칩 메모리 용량을 물리적으로 초과한다. 따라서 예산 24를 초과하는 시점부터는 필연적으로 캐시 미스(Cache Miss)가 발생하여, 연산 코어의 동작 시간보다 메인 메모리(RAM)에서 데이터를 가져오거나 쓰는 데 대부분의 시간을 허비하게 된다.

결론적으로, 서버 환경에서 효과적인 큰 드래프트 예산을 모바일 환경에 그대로 적용하는 것은 오히려 전체 추론 속도를 저하시키는 비효율성을 야기하며, 제한된 드래프트 예산 내에서 평균 수락 길이를 극대화할 수 있는 최적의 트리 구조를 탐색하는 것이 필수적이다.

또한, 드래프트 모델은 단일 트랜스포머 레이어로 구성되어 있어 타겟 모델에 비해 크기가 매우 작으며, 트리 생성 과정에서 사용되는 배치 크기가 4를 넘지 않기 때문에 배치 크기가 증가함에 따라 연산 지연 시간이 과도하게 커지는 현상은 발생하지 않는다.

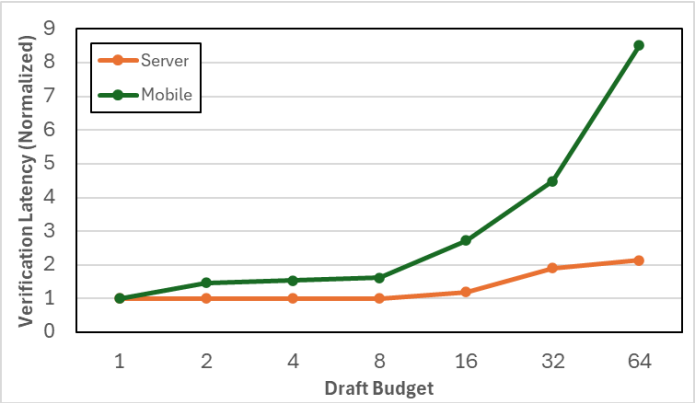

그림 1. 드래프트 예산별 검증 지연시간 비교. 모든 지연시간은 드래프트 예산 1을 기준으로 정규화 한 결과임.

3-2. 비용 함수(Cost Function)와 최적화 목표

추측 디코딩의 전체(end-to-end) 생성 속도를 평가하는 핵심 지표는 초당 생성 토큰 수(Tokens per Second)이며, 이는 토큰 당 지연시간(Per-Token Latency)과 역수 관계에 있다. 드래프트 트리의 예산 n은 트리의 깊이 D, 트리의 너비 W, 한 노드에서 확장되는 자식 노드의 수를 나타내는 트리의 분기 계수(Branching Factor, BF)에 의해 결정된다. 본 연구에서는 최적의 트리 구조를 찾기 위한 비용 함수로 토큰 당 지연시간을 사용하며, 이는 다음 수식으로 정의된다.

$$T_{drafting} = T_d(B) \times D \approx T_d(1) \times D \quad (1)$$

$$T_{verification} = T_{target}(n) \quad (2)$$

$$Cost = \frac{(T_{drafting} + T_{verification})}{\tau} = L_{per\ token} \quad (3)$$

$T_{drafting}$ 은 드래프트 트리 생성 시간을 나타낸다. T_d (B)는 드래프트 모델이 트리의 한 깊이에서 B 크기의 배치를 연산하는 시간이며 트리의 생성 과정에서 D번 연산이 수행된다. 드래프트 모델은 배치 크기의 변화가 지연시간에 미치는 영향이 미미하므로, 드래프트 트리 생성 시간은 단일 배치 연산 시간 T_d (1) 와 D에 비례한다고 근사(Approximate)할 수 있다. $T_{verification}$

은 드래프트 예산 n에 따른 검증 시간, τ는 평균 수락 길이, $L_{per\ token}$은 토큰 당 지연시간을 나타낸다. 식 (3)을 보면 트리의 깊이나 예산을 늘리면 분자인 드래프트 생성 및 검증 시간이 증가하지만, 동시에 분모인 평균 수락 길이도 증가하는 경향이 있다. 따라서 본 논문에서는 식 (3)의 비용 함수를 최소화하는 트리 구조를 찾는 것을 목표로 설정하였다.

3-3. 그리드 서치를 통한 최적 트리 구조 탐색

본 논문에서는 Speed-up Ratio, 메모리 사용량 측면에서 SOTA(state-of-the-art) 성능을 보이는 트리 기반 추측 디코딩 기법인 EAGLE[3]에서 제안하는 트리 구조를 Baseline으로 설정하였다. 최적화된 트리 구조를 효율적으로 탐색하기 위해, 먼저 Baseline인 EAGLE의 정적 트리(그림 2-(a))에서 토큰 수락 경향성을 분석하였다. 그 결과, 드래프트 모델의 확신도가 가장 높은 주 경로와 그 인접 노드에서 생성된 토큰이 전체 수락 토큰의 79% 이상을 차지하는 뚜렷한 패턴을 발견하였다.

따라서, 확률이 높은 경로는 깊게 탐색하고 나머지 경로는 너비를 줄이는 비대칭적 트리 구조들을 후보군으로 설정하였다. 탐색 공간(Search Space)으로 $3 \leq D \leq 5$, $1 \leq W \leq 8$, $1 \leq BF \leq 3$, $3 \leq n \leq 25$로 정의하였으며, 3-2절에서 정의한 비용 함수를 기반으로 그리드 서치(Grid Search) 수행하였다.

탐색 결과, 그림 2-(b)와 같은 모바일 환경에 최적화된 최종 트리 구조를 도출하였다.

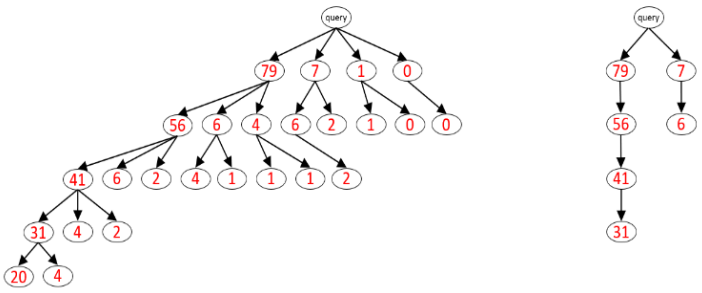

그림 2. (a) EAGLE-1의 정적 트리 구조(Baseline),
(b) 모바일 환경에서 효율적인 트리 구조(Proposed).

4. 실험

4-1. 실험 설정

본 논문에서 제안하는 정적 트리 구조의 효율성을 검증하기 위해 GSM-8K 데이터셋을 활용하였다. 모든 실험은 Qualcomm Snapdragon 8 Elite AP(Oryon-L x 2, Oryon-M x 6 CPU, Adreno 830 GPU)와 12GB RAM을 탑재한 Samsung Galaxy S25 스마트폰을 사용하여 수행되었다. 추측 디코딩 알고리즘은 EAGLE을 llama.cpp 프레임워크 상에 구현하여 실험을 진행하였다. 실험에 사용한 모델은 타겟 모델로 4비트 양자화된 Vicuna-7B, 드래프트 모델로 4비트 양자화된 EAGLE Layer(Vicuna-7B)를 사용하였다.

4-2. 실험 결과

먼저, 추측 디코딩을 적용하지 않은 자기 회귀 방식의 추론 속도는 11.49 tps로 측정되었다. Baseline인 EAGLE의 정적 트리 구조를 적용한 실험에서, 추론 속도는 8.99 tps에 그쳤다. 이는 자기 회귀 방식보다 약 21.8% 느린 수치로, 3-1절에서 분석한 바와 같이 모바일 환경에 최적화되지 않은 큰 드래프트 예산이 심각한 검증 병목을 유발함을 보여준다. 반면, 본 논문에서 제안하는 모바일 특화 정적 트리를 적용한 실험 결과 추론 속도는 19.18 tps을 기록하여 자기 회귀 방식 대비 약 1.67배, Baseline 대비 약 2.13배 향상된 성능을 달성했다. Baseline 보다 평균 수락 길이는 4.06에서 3.22로 다소 감소하였으나, 드래프트 트리 검증 시간이 훨씬 더 큰 폭으로 감소하여 전체 처리량(tps)을 성공적으로 극대화하였다.

표 1. 제안하는 트리 구조와 Baseline의 성능 비교

	Draft Budget	Drafting Latency (ms)	Verification Latency (ms)	평균 수락 길이	초당 토큰 생성 수 (token per second)
기존	25	52.34	392.12	4.06	8.99
개선	6	47.1	118.64	3.22	19.18

5. 결 론

본 논문은 모바일과 같은 하드웨어 자원이 제한된 환경에서 LLM의 추론속도를 가속하기 위한 트리 기반 추측 디코딩 기법의 효율적 트리 구조를 제안했다. 본 연구는 모바일 기기의 하드웨어 성능 제약을 핵심적인 제약 조건으로 인식하고 이를 해결하는 데 중점을 두었다. 실험 결과, 제안된 트리 구조는 과도한 후보 생성으로 인한 연산 자원 낭비를 최소화하면서도 높은 검증 성공률을 유지하여, 기존의 트리 기반 추측 디코딩 기법에 비해 우수한 추론 가속 성능을 보였다.

참 고 문 헌

[1] Yaniv, Leviathan., et al. Fast Inference from Transformers via Speculative Decoding. Proceedings of the 40th International Conference of Machine Learning, Article No.: 795 (pp. 19274-19286). (2023).

[2] Xupeng, Miao., et al. Specinfer: Accelerating Large Language Model Serving with Tree-based Speculative Inference and Verification. Proceedings of the 29th ACM International Conference of Architectural Support for Programming Languages and Operating Systems, Volume 3 (pp. 932-949). (2024).

[3] Yuhui, Li., et al. EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty. arXiv preprint (arXiv:2401.15077). (2024).