LLM in a flash- Efficient Large Language Model Inference with Limited Memory

Keivan Alizadeh, Iman Mirzadeh* , Dmitry Belenko* ,
S. Karen Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, Mehrdad Farajtabar

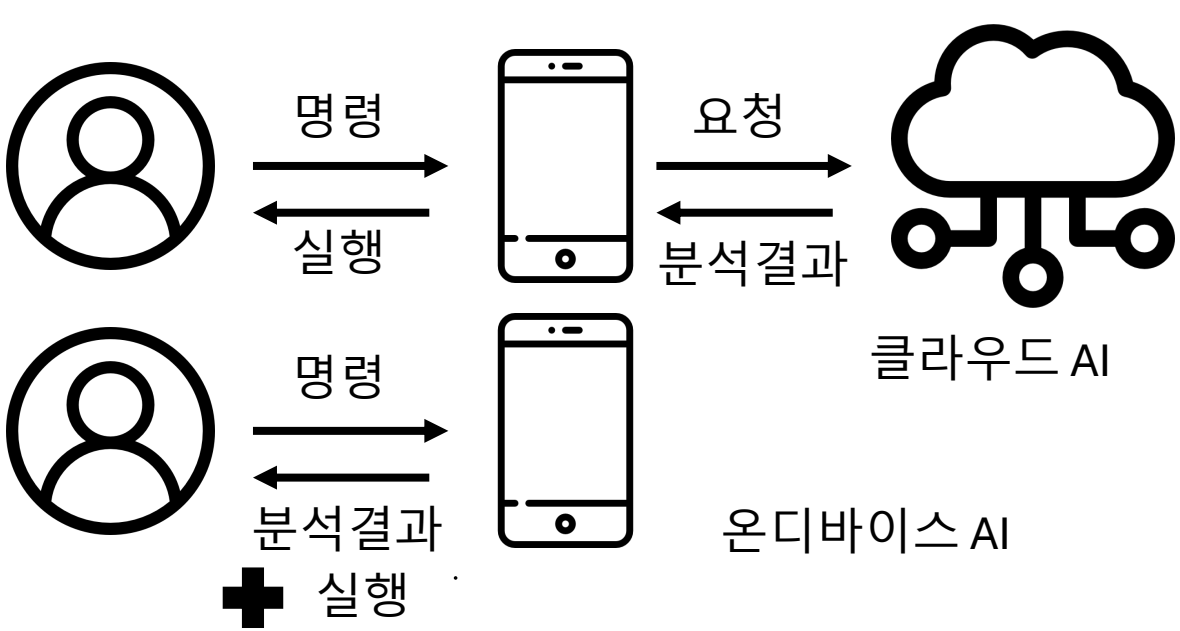
2023.12.12

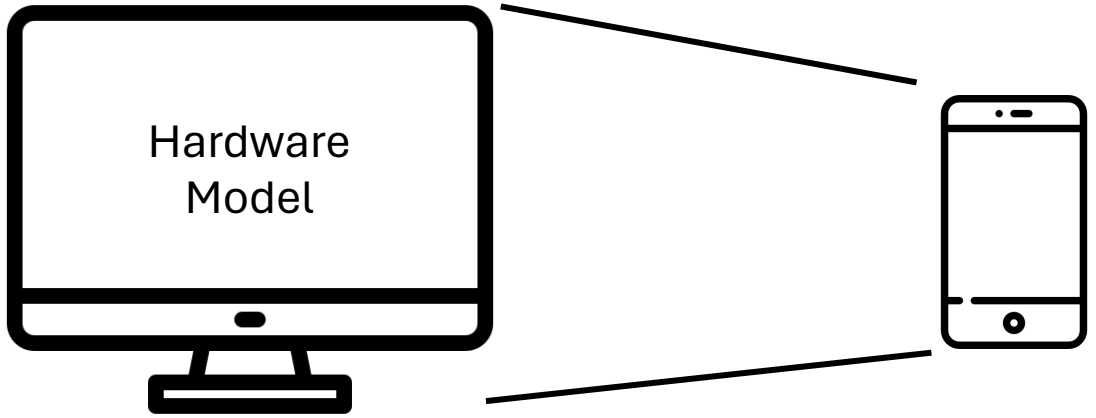
발표자 : 김산

Introduction

온디바이스에 LLM을 적용하기

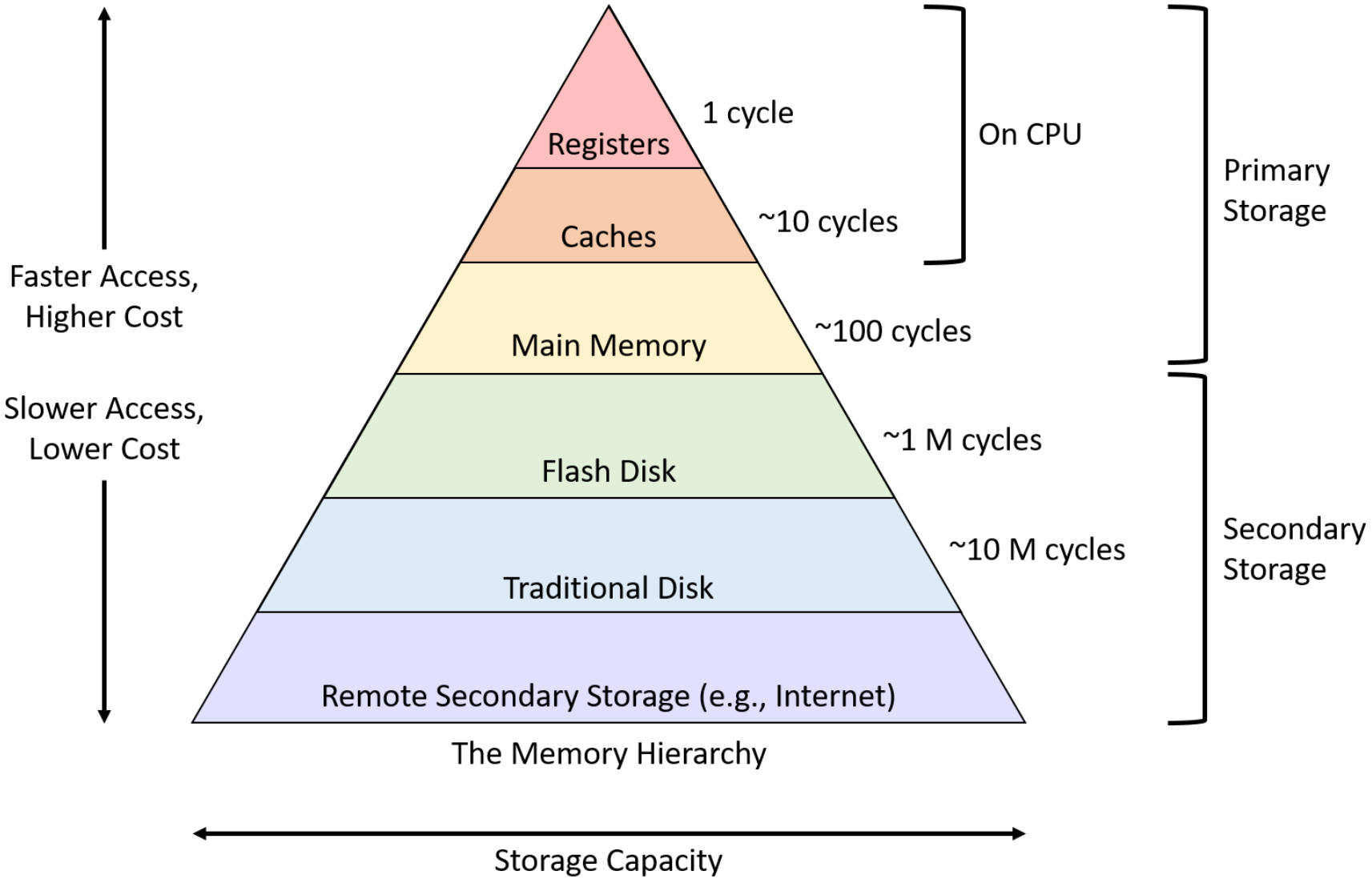
- **개인정보 보호** : 모든 데이터를 디바이스에서 처리해 개인정보가 외부로 유출될 위험이 적다.
- **오프라인 사용** : 디바이스 자체에서 작업이 수행되기 때문에 인터넷 연결 없이도 사용할 수 있다.
- **실시간 처리** : 서버와의 통신에 의존하지 않고 디바이스 내에서 처리하기 때문에 실시간 데이터 분석 및 처리가 가능하다.
- **하드웨어 한계** : LLM은 CPU, GPU, 메모리 등 많은 자원을 필요로 한다. 핸드폰과 같은 디바이스는 LLM을 처리하기에 충분한 자원을 가지고 있지 않다.
- **모델 최적화** : 환경이 다른 만큼 디바이스에서 LLM을 실행하기 위해서는 모델을 작게 최적화할 필요가 있다.





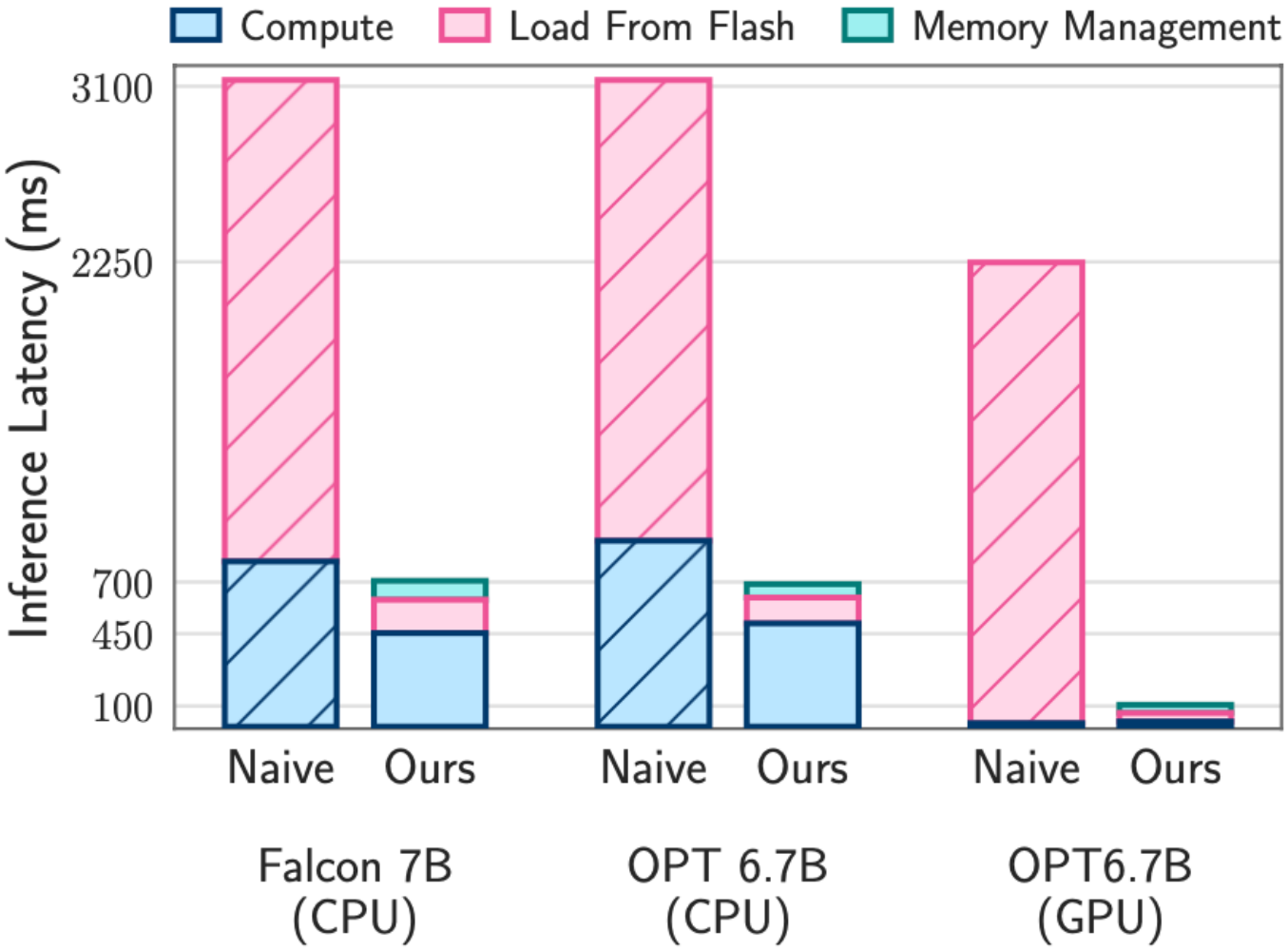
Introduction

Memory Hierachy



DRAM + Flash Memory

Introduction



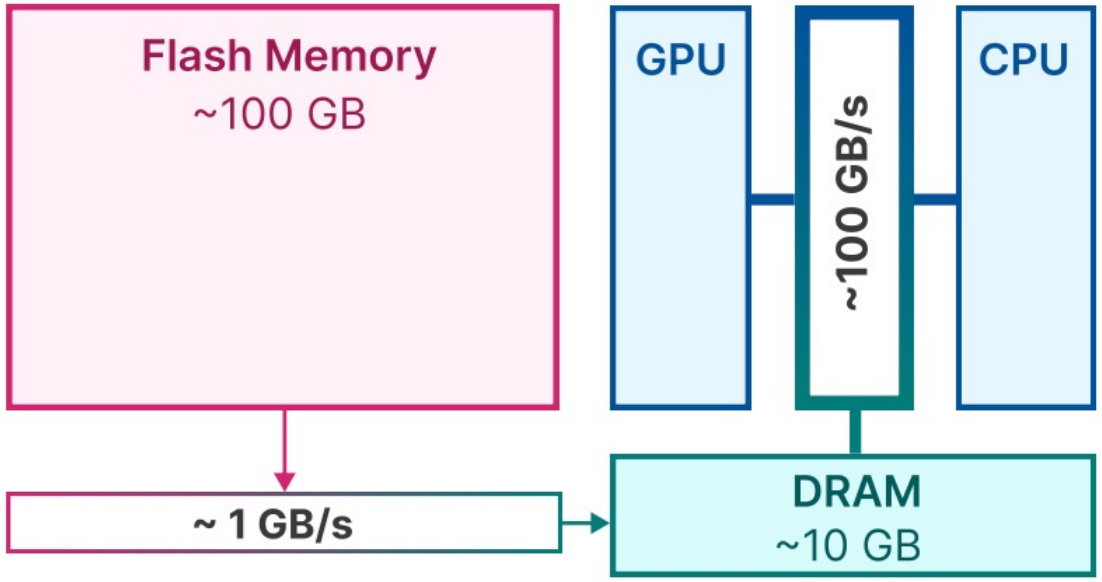
M1 Max / NVIDIA GeForce RTX 4090 환경에서 수행

7B모델을 BF16을 이용해 표현하면 얼마의 메모리가 필요할까?

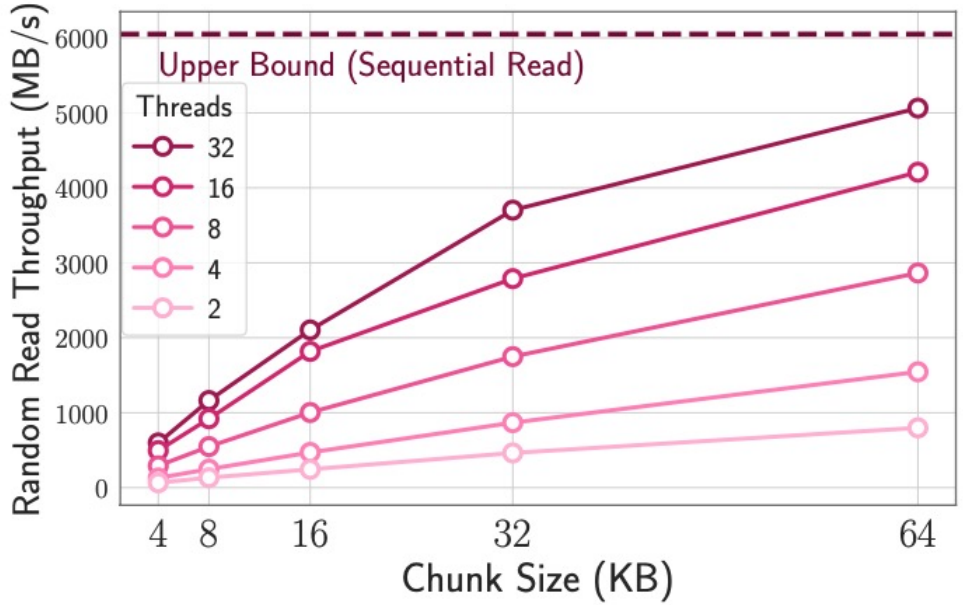
→ 7*10^9*2Byte = 14GB

사용가능한 DRAM 보다 모델의 가중치가 더 크기 때문에 모델의 전체 가중치를 Flash memory에 저장해 로드해야 함.

Introduction



(a) Bandwidth in a unified memory architecture



(b) Random read throughput of flash memory

LLM을 돌리기 위해서는 충분한 용량(Flash Memory)과 속도(DRAM)가 필요하다.

⟶ 충분한 Bandwidth 필요

- Chunk Size가 커져도 여전히 느린 처리속도

1. 필요한 부분을 작은 Chunk로 읽기 보다 필요 이상의 데이터를 큰 Chunk로 읽고 버리는 것이 더 나을 수 있음
2. Flash Memory의 병렬 읽기 특성을 활용하여 읽기 연산을 병렬화 함

위의 이유로 이전에 DRAM을 사용했지만 본 논문에서는 **Flash Memory와 함께 사용**하는 방법을 제시

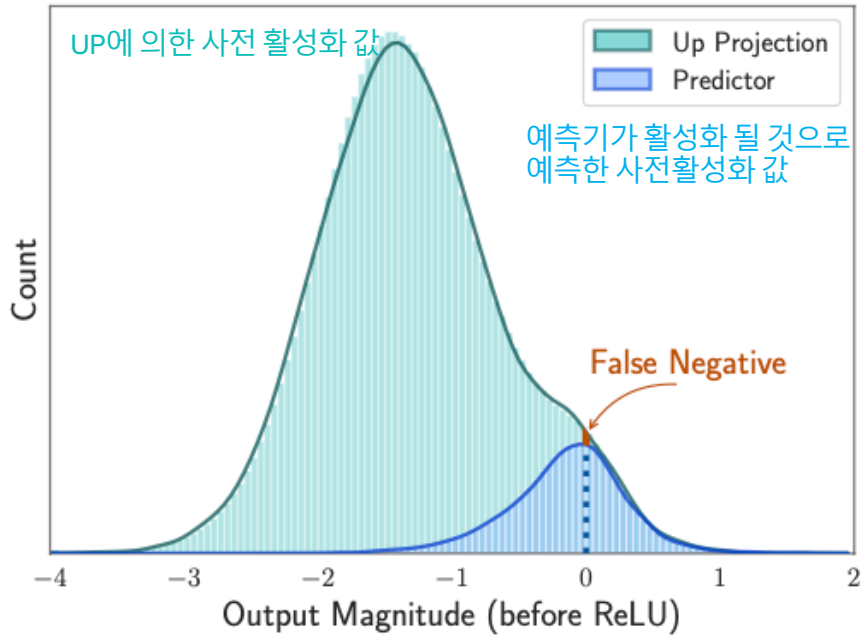
Load From Flash

3가지 방법을 사용하였다.

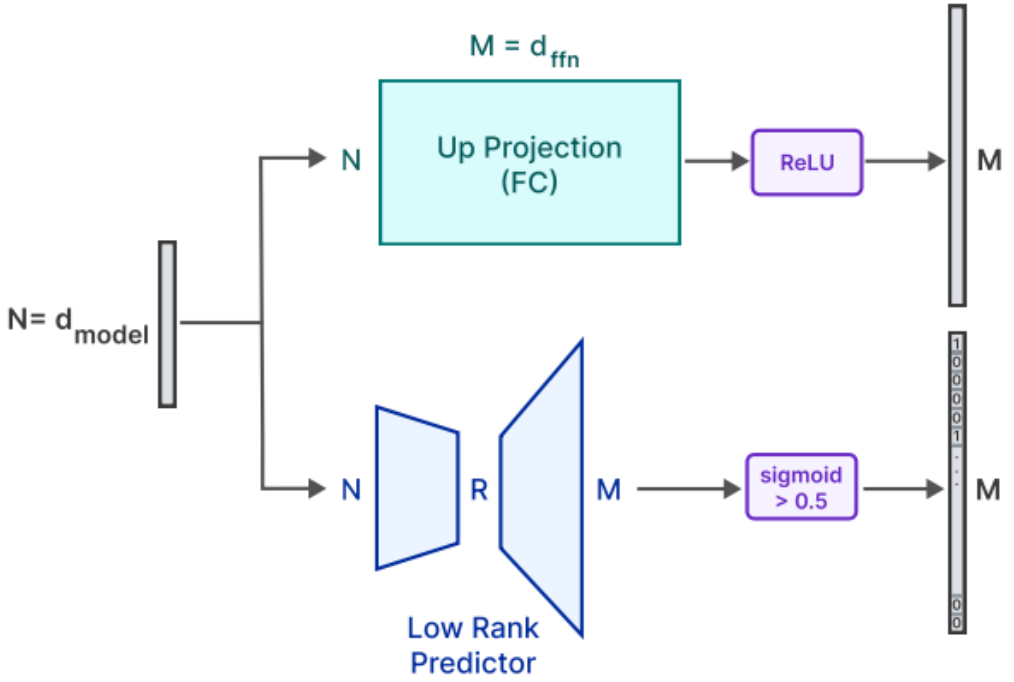
- Predictor
- Sliding window
- Ro—Column Bundling

Load From Flash

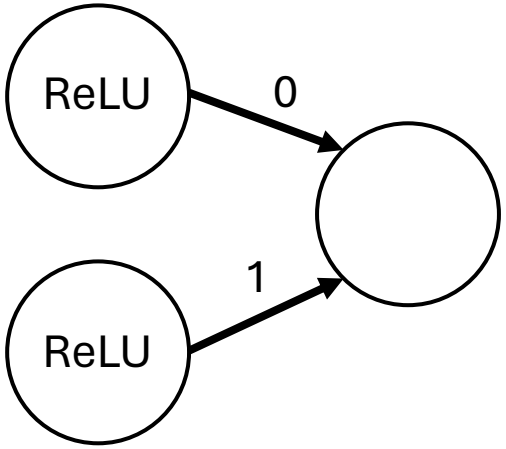
- Predictor



(a) predictor vs relu



(b) low rank predictor

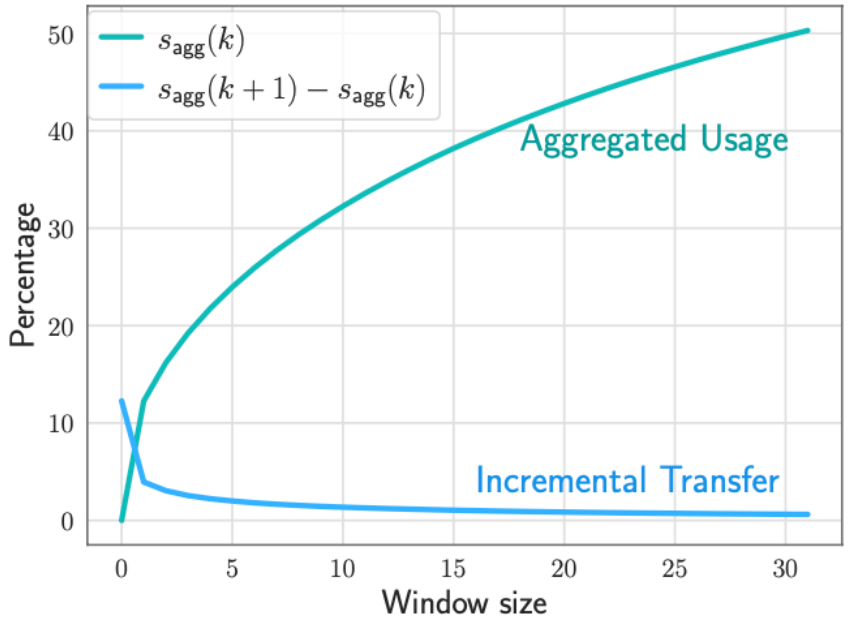


1. FFN의 특성을 활용하여 데이터 전송 속도를 개선
2. Low-Rank Predictor를 사용해 활성화 여부 예측
3. ReLU 활성함수에서 활성화된 부분만을 Load
4. 이 방법은 크고 작은 모델 모두에서 효과적으로 사용할 수 있다

Load From Flash

- Sliding window

window size에 따른 신경 사용량의 변화



(a) aggregated neuron use

(b) sliding window

사용하지 않는 뉴런은 unload하고 사용하는 뉴런은 load하여 전송 데이터 양을 감소

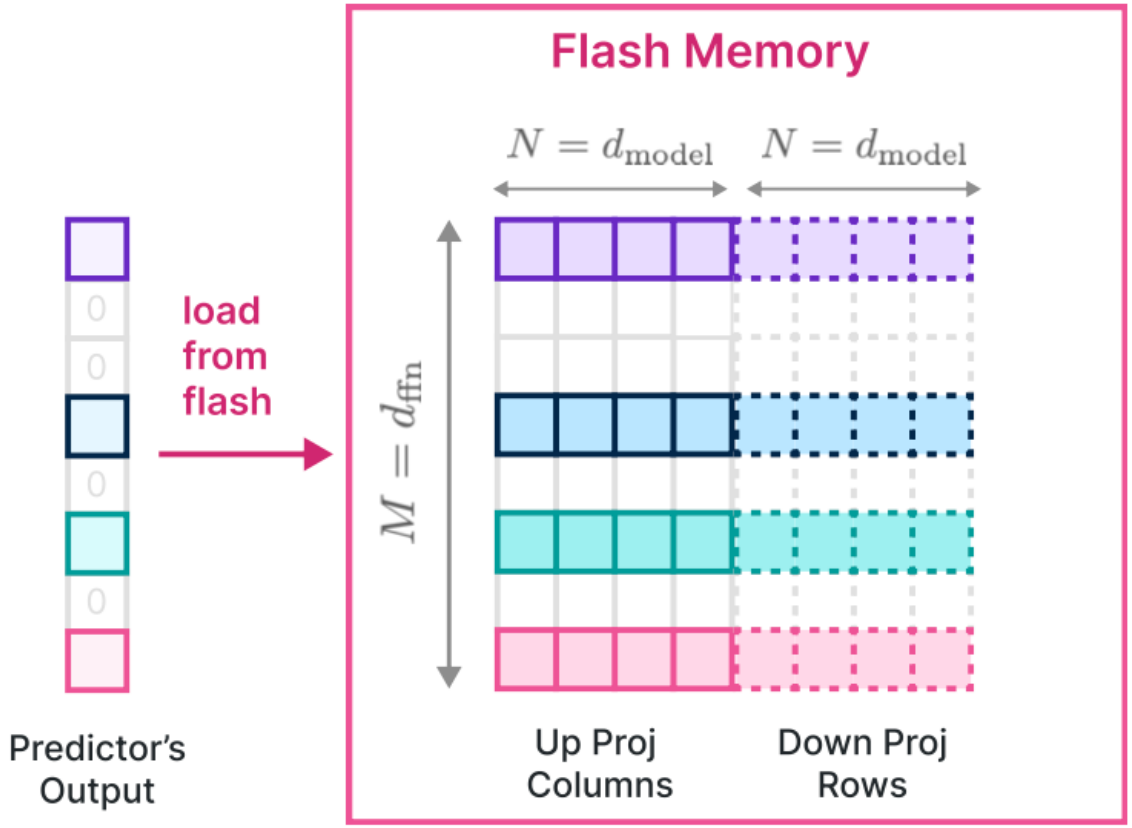
이동하면서 자주 사용되는 부분만 사용

⟶ I/O 요청수를 줄이고자 함.

1. Flash ⟶ DRAM으로 전송해야 하는 데이터 양 감소
2. 메모리 절약

Load From Flash

- Row-Column Bundling

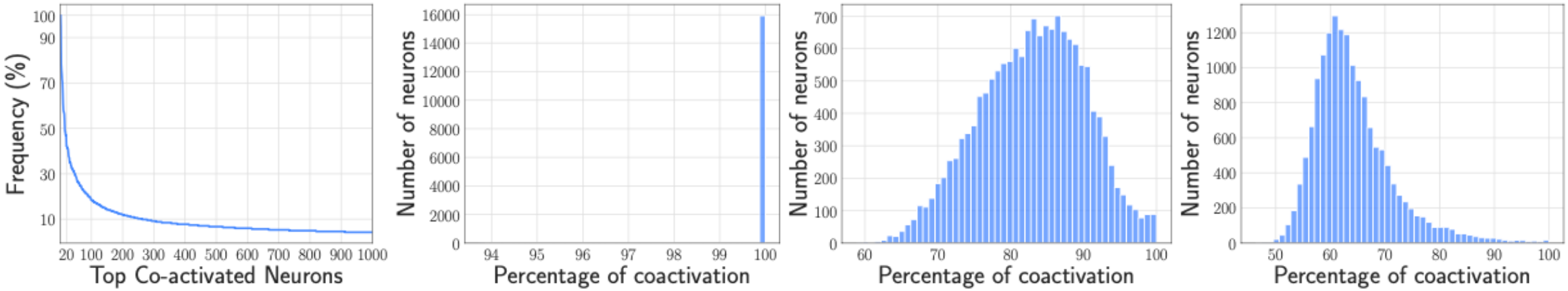


- Up project : 여러 행을 묶어서 처리. 데이터를 전방향으로 전달
- Donw project : 여러 열을 묶어서 처리. 데이터를 역방향으로 전달.

행열을 함께 Flash Memory에 저장해 데이터를 더 큰 Chunk로 읽을 수 있도록 한다.

- 동시에 여러 데이터 로드 : 여러 데이터를 함께 처리하여 데이터 처리량 향상
- Flash Memory의 효율적 사용 : 큰 Chunk로 읽을 때 더 빠르고 병렬 처리가 가능한 Flash Memory의 특성을 살려 Flash Memory를 효율적으로 사용

Load From Flash



(a) coactivation intensity (b) Closest friend (c) 4th closest friend (d) 8th closest friend

(a) 공동으로 활성화된 뉴런들의 빈도 그래프 : 특정 뉴런과 함께 활성화되는 뉴런 그룹이 있음을 보여줌. 활성화 빈도가 높을수록 숫자가 감소하는 것을 볼 수 있음. ⟶ 가장 자주 활성화 하는 뉴런을 파악하는 것이 중요

(b) 가장 가까운 뉴런의 공동 활성화 빈도 : 거의 항상 함께(95%) 활성화 되는 것을 볼 수 있음 ⟶ Row-Column Bundling의 근거(공동 활성화 기반 번들링을 했을 때 특정 뉴런이 자주 활성화되어 여러 번 로드되는 문제 발생. Row-Column Bundling은 공동 활성화 패턴을 고려하지 않고 사용할 수 있어 효율을 높일 수 있음.

(c) 4번째로 가까운 친구의 활성화 빈도

(d) 8번째로 가까운 친구의 활성화 빈도

Results

Configuration				Performance Metrics			
Hybrid	Predictor	Windowing	Bundling	DRAM (GB)	Flash→ DRAM(GB)	Throughput (GB/s)	I/O Latency (ms)
✗	✗	✗	✗	0	13.4 GB	6.10 GB/s	2130 ms
✓	✗	✗	✗	6.7	6.7 GB	6.10 GB/s	1090 ms
✓	✓	✗	✗	4.8	0.9 GB	1.25 GB/s	738 ms
✓	✓	✓	✗	6.5	0.2 GB	1.25 GB/s	164 ms
✓	✓	✓	✓	6.5	0.2 GB	2.25 GB/s	87 ms

1. 데이터 전송량이 0.2GB로 감소했다.
2. I/O Latency가 87ms로 빨라졌다

Conclusion

1. on-device에서 LLM을 사용하기 위해 Flash Memory와 DRAM을 함께 사용하는 방식을 제안
2. Predictor / Sliding window / Ro-Column Bundling을 사용
3. I/O요청수와 Latency를 줄일 수 있었다.