LLM INFERENCE

Inference Engineering & AI 반도체

이진원 · HyperAccel CTO

2026

“

We used to talk for years at OpenAI about
when we were going to get through the compute crunch,
when we were going to build enough compute
that we wouldn’t be so strapped.

I don’t think we ever
get out of it.

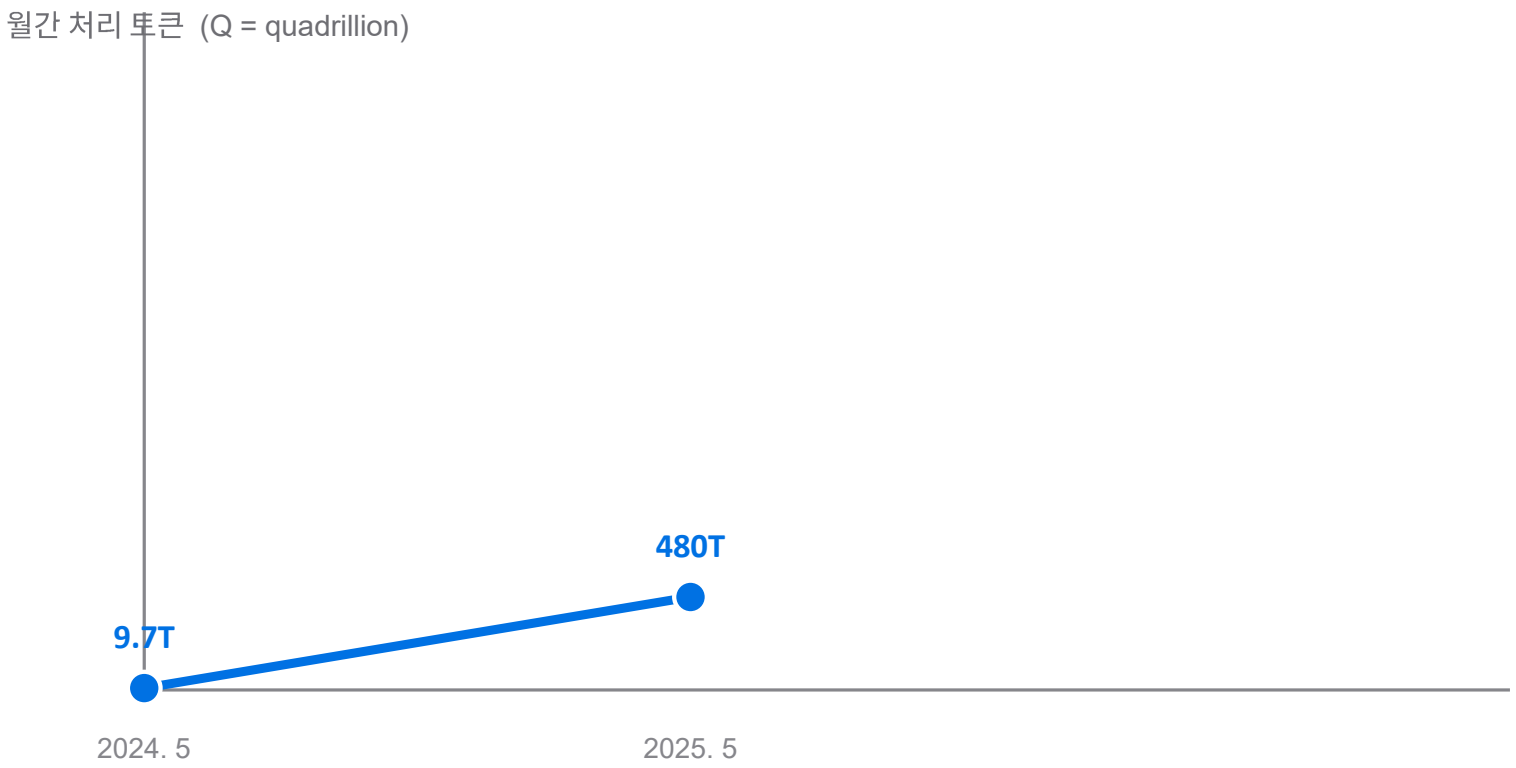
Sam Altman

OpenAI Forum — “Sam Altman on Building the Future of AI”, 2026. 4. 6.

수요

Explosive Growth in AI Demand

Google 한 곳이 한 달에 처리하는 토큰



50×

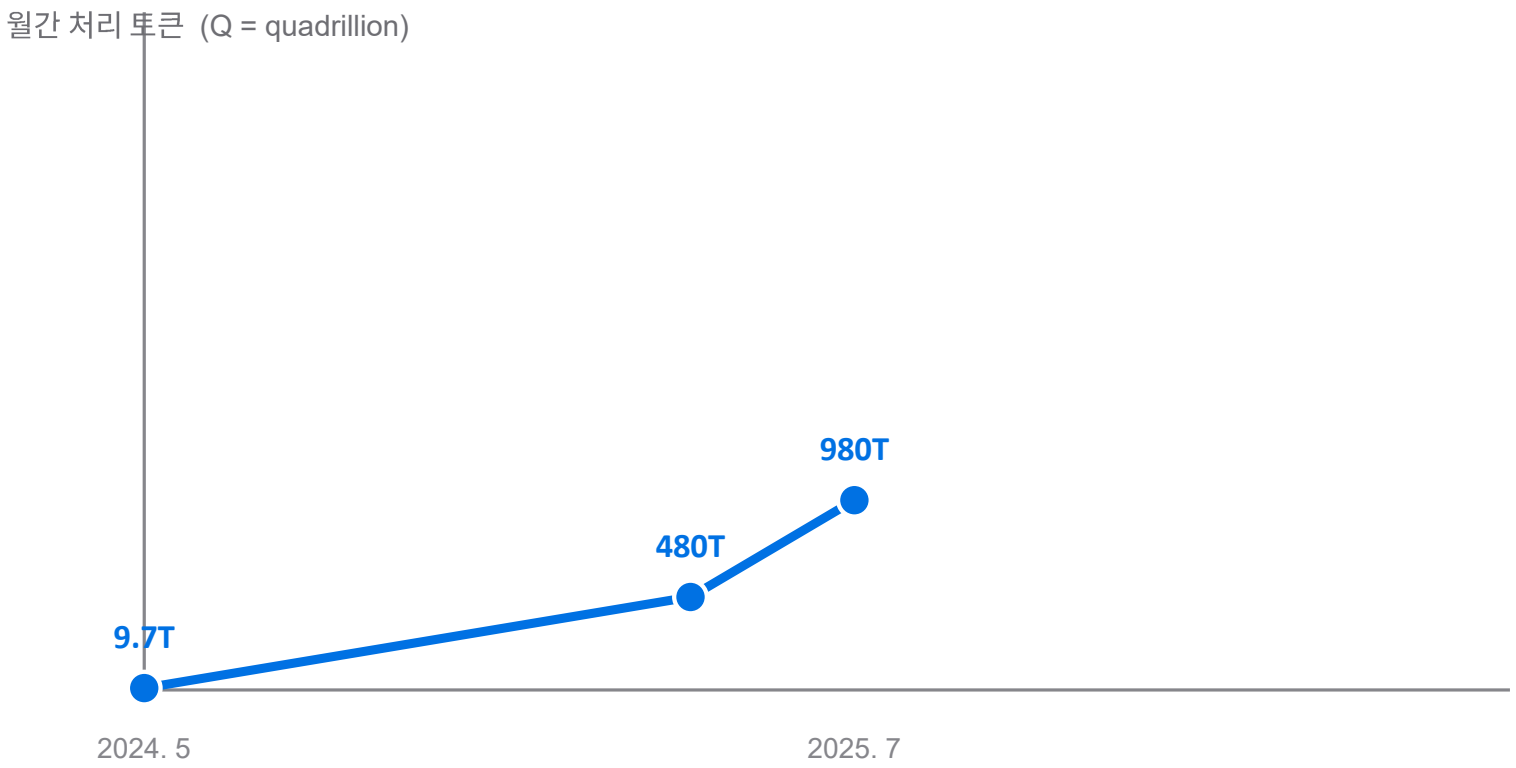
12개월 만에 · 9.7T → 480T

출처 : Sundar Pichai — Google I/O 2024·2025·2026 및 분기 실적발표(월 3.2 quadrillion · 전년비 7배) · OpenRouter 주간 25T 토큰(2026-05) · y축 0~3.5Q 고정.

수요

Explosive Growth in AI Demand

Google 한 곳이 한 달에 처리하는 토큰



2×

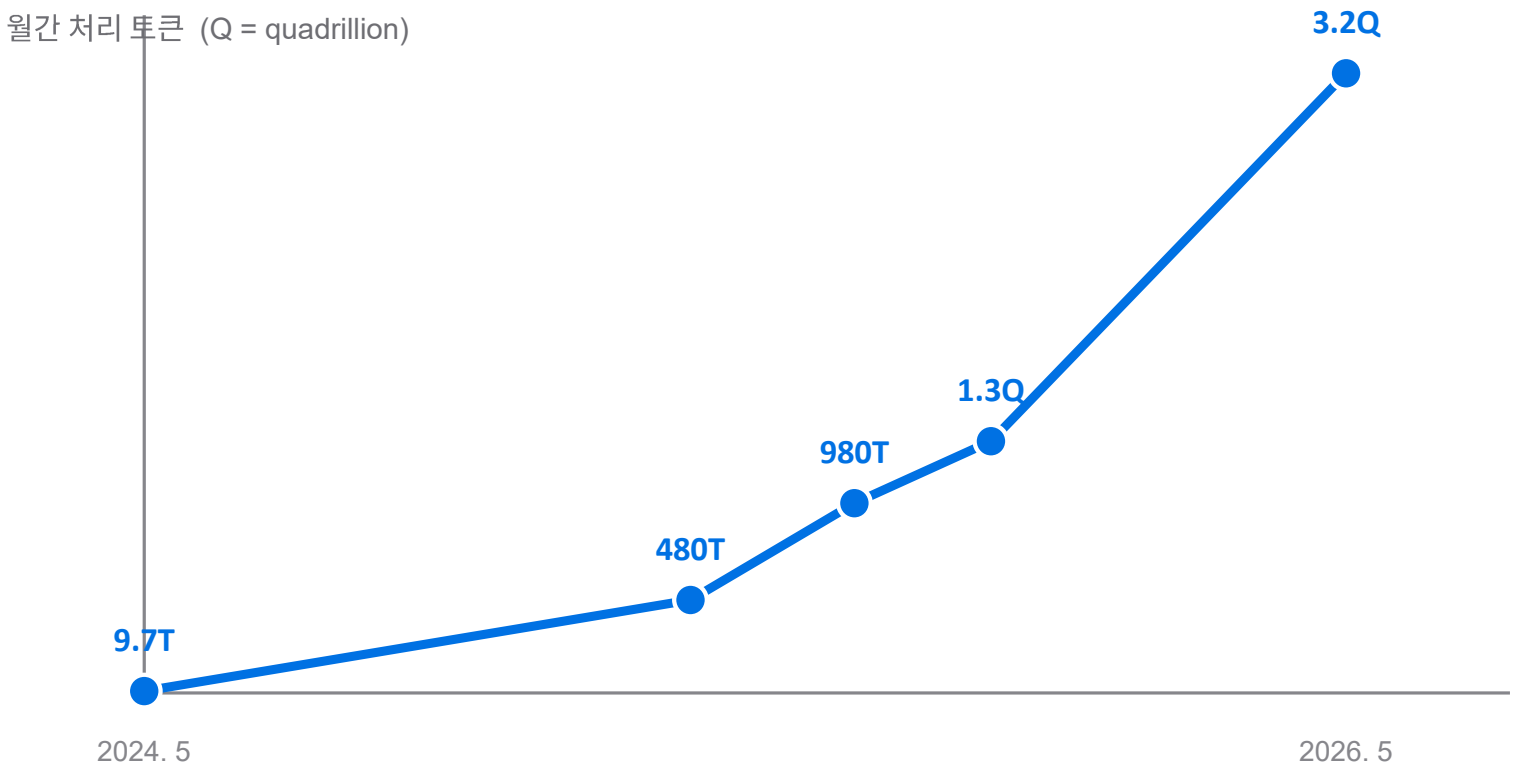
다시 두 달 만에 · 480T → 980T

출처 : Sundar Pichai — Google I/O 2024·2025·2026 및 분기 실적발표(월 3.2 quadrillion · 전년비 7배) · OpenRouter 주간 25T 토큰(2026-05) · y축 0~3.5Q 고정.

수요

Explosive Growth in AI Demand

Google 한 곳이 한 달에 처리하는 토큰



330×

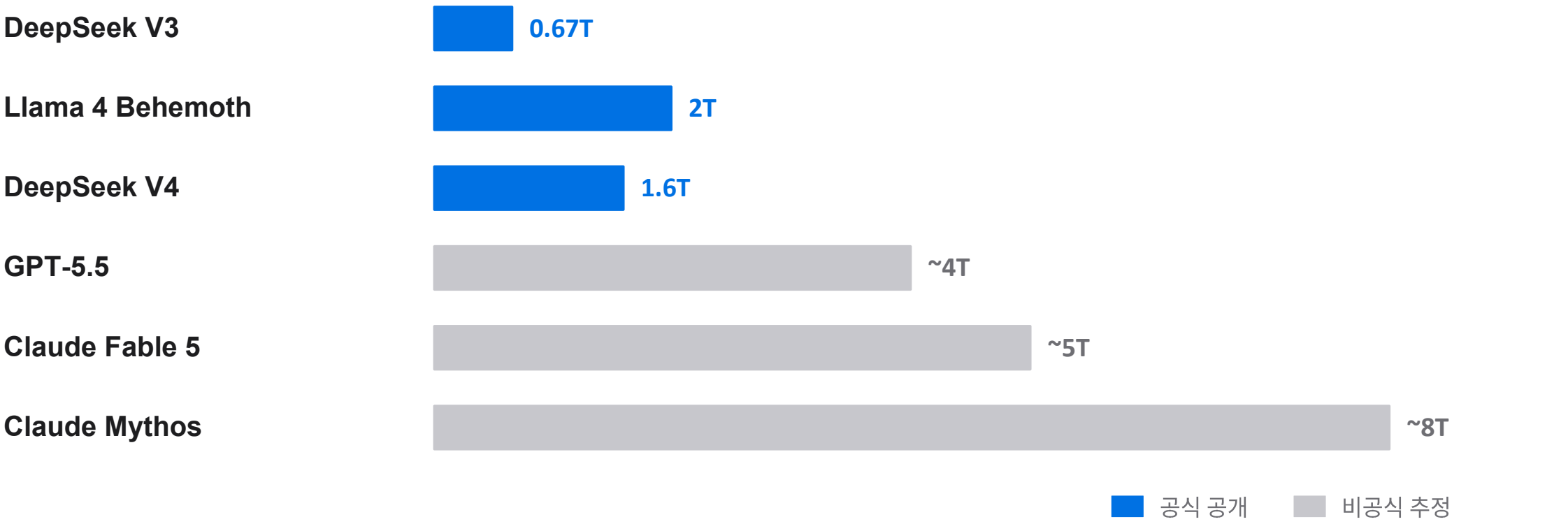
25개월 만의 증가 · 전년 대비로도 7배

출처 : Sundar Pichai — Google I/O 2024·2025·2026 및 분기 실적발표(월 3.2 quadrillion · 전년비 7배) · OpenRouter 주간 25T 토큰(2026-05) · y축 0~3.5Q 고정.

수요

그리고 모델이 다시 커진다

「스케일링은 끝났다」던 논쟁이 무색하게 파라미터가 다시 조 단위로 감



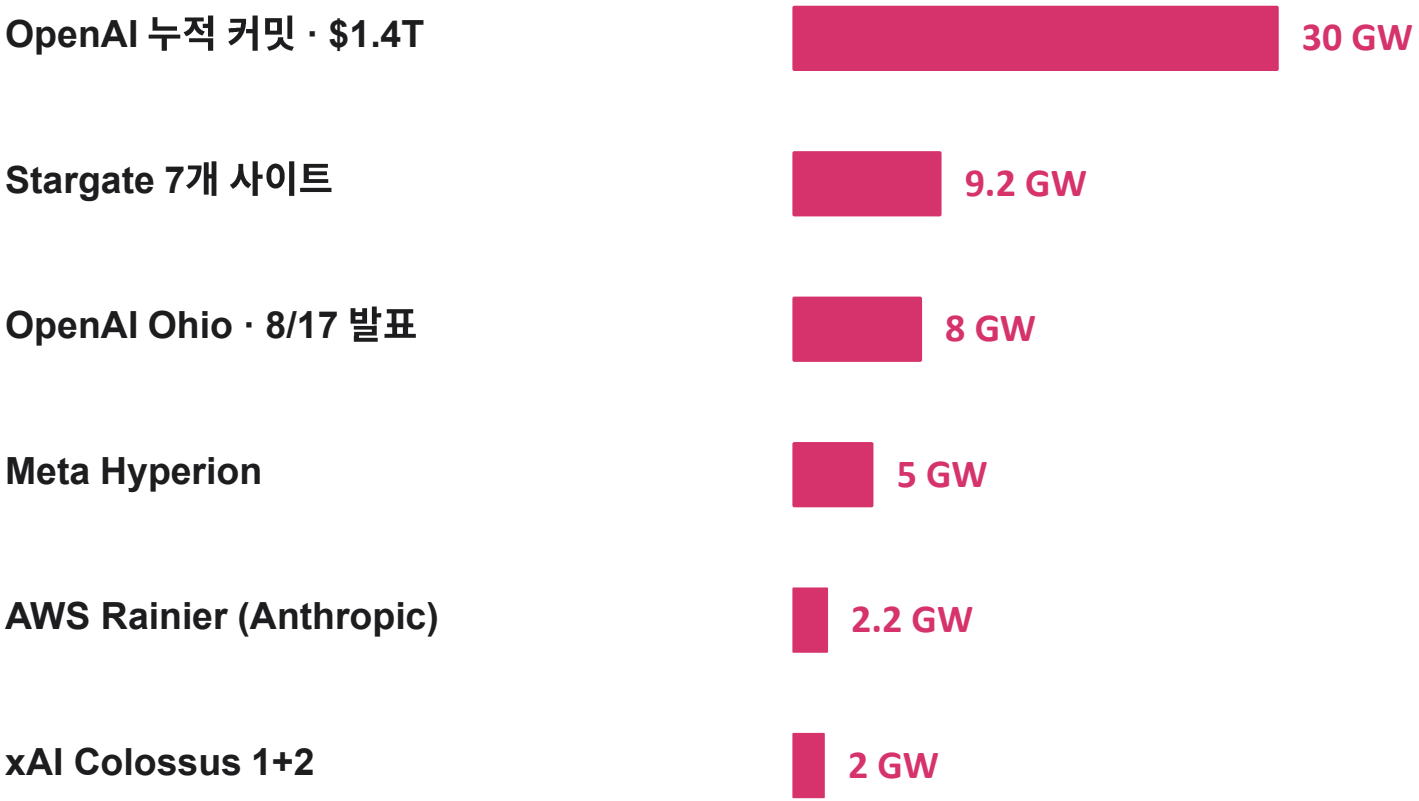
모델이 커지면 가중치도 KV도 같이 커진다 — 앞에서 본 용량·대역폭 문제가 그대로 확대된다

DeepSeek V4·Llama 4는 공식 수치 · Fable 5 / Mythos는 파라미터 비공개(FT 추정 ⚠) · GPT-5.5는 리크 · Musk(2026-01) · Anthropic-AWS(2026-04).

인프라

GW 데이터센터의 시대

발표된 AI 데이터센터 캠퍼스 전력



$6.7T

2030년까지 DC 투자 · McKinsey 2025 · IT 장비 $3.3T가 최대 항목

$630~725B

빅4의 2026년 capex

투자의 절반

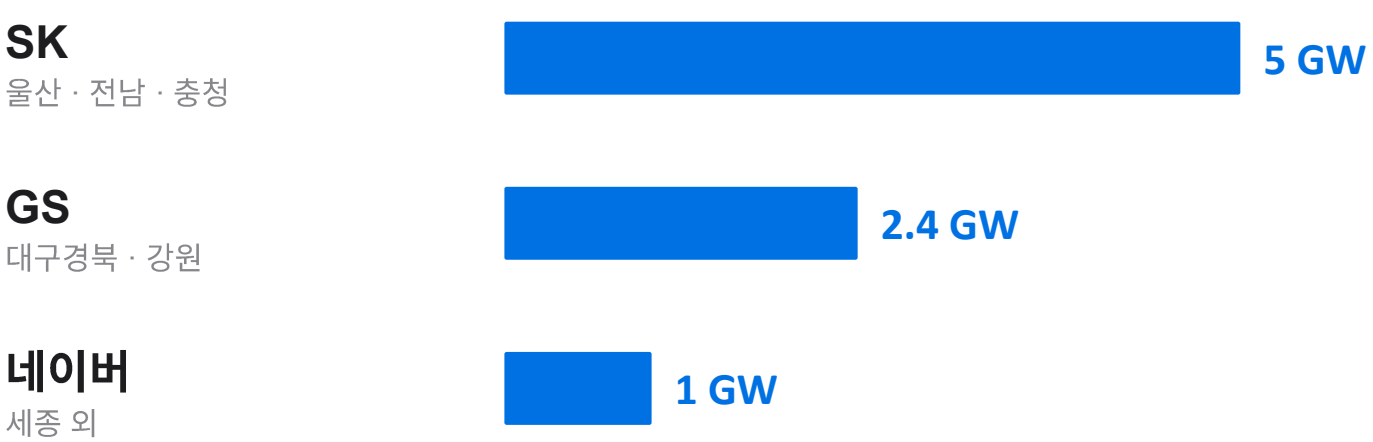
칩과 메모리로 감

출처 : OpenAI·Meta·AWS·xAI 공식 발표 · Epoch AI 집계(2026-04 : Stargate 가동은 ~0.3 GW) · Ohio 8 GW는 CNBC 2026-08-17 · McKinsey(2025-04).

인프라 · 한국

한국도 GW 데이터센터 경쟁에 참여

2026. 6. 29. 3대 메가프로젝트 국민보고회 — AI 데이터센터 1단계 8.4 GW



1단계 합계	**8.4 GW**	2029년 · 투자 약 550조원
2단계	**SK 5 → 15 GW (2035) · 총 18.4 GW**	

14.37 GW

2030년 국내 목표 · 아시아태평양 58 GW의 25%

정부가 함께 명시

「NPU 등 국산 AI 반도체를 필두로 한 AI 추론 시장 선점」

전제도 정부가 짚음

「AI 경쟁력의 핵심, 안정적 전력·용수 공급」

출처 : 산업통상자원부 「대한민국 대도약 3대 메가프로젝트 국민보고회」(2026-06-29) 참고자료 · 2030년 14.37 GW·아태 58 GW 및 지역 배분은 전자신문(2026-06-29) 보도.

인프라

병목은 전력이다

연산도 메모리도 아니고, 꽂을 전력이 없어서 멈춤

130 → 1,000 kW
랙 하나의 전력
GB200 → 차세대 로드맵

5년
신규 발전소 계통 연결까지
LBNL 중위 61개월

58.3%
수도권 데이터센터 전력 본심사 탈락률

$1.4T
OpenAI 누적 커밋
30 GW

「GPU를 꽂을 전력이 없어 재고로 쌓여 있다」 — Satya Nadella · Microsoft CEO · 2025. 10.

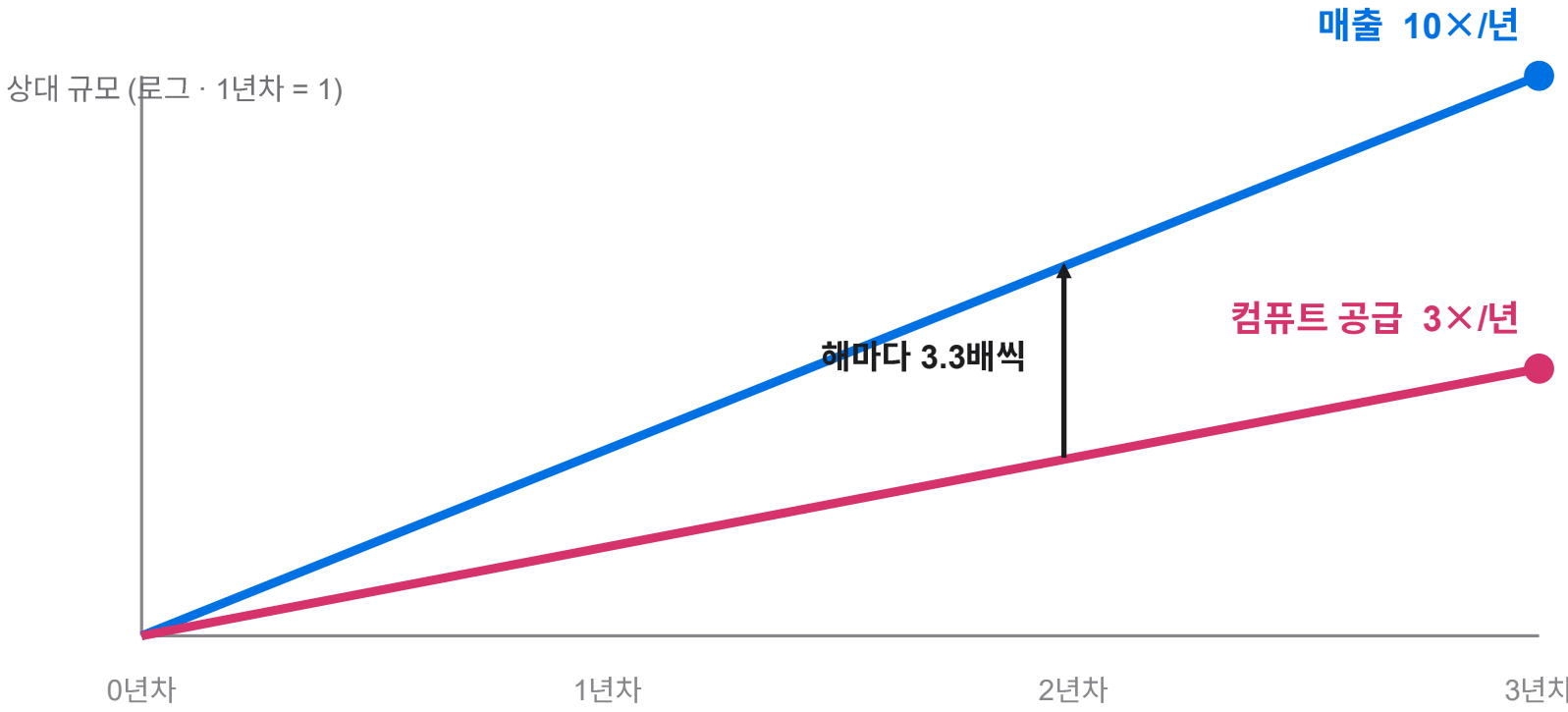
전력이 게이트라면 칩의 성적표는 FLOPS가 아니라 tokens / Watt 와 tokens / $ 다

출처 : Nadella — BG2 팟캐스트(2025-10-31) · LBNL 「Queued Up」 2026(접속 신청부터 상업운전까지 중위 61개월, 대기열 ~2,061 GW) · 랙 전력은 Bloomberg Intelligence(2026-01).

컴퓨트 가격

매출은 해마다 10배, 컴퓨트는 3배

Anthropic 3년 연속 매출 10× · 프론티어 랩이 확보하는 연산은 연 3배



3.3×

해마다 벌어지는 간극

▲ 37배

3년이면 이만큼

돈이 없어서 못 사는 것이 아니다 — 살 수 있는 컴퓨트가 그만큼 없다

출처 : Anthropic Series G 공시(2026-02-12) 「3년 연속 매출 10×」 · ▲ 컴퓨트 연 3배는 업계 공급 증가율(Epoch · Dario 2026-02) 기준.

컴퓨트 가격

간극을 메우는 세 가지 방법

해마다 3.3배씩 벌어지는 간극 — 매출 연 10× ÷ 컴퓨트 공급 연 3×

거의 소진

① 추론 마진을 올린다

38% → 70%+

이미 두 배 올렸음. 상한은 90% 안팎 — 거의 소진.

거의 소진

② 추론 비중을 올린다

25% → 50%+

학습을 멈출 수는 없음. 비중에도 천장이 있음 — 거의 소진.

진행 중

③ 컴퓨트 단가가 오른다

+40% · +48%

H100 1년 계약 · B200 현물. 천장이 없는 유일한 길 — 진행 중.

앞의 둘이 한계면 남는 길은 하나 — GPU 시간의 값이 오르고, 그 값은 토큰당 원가로 돌아온다

출처 : Anthropic Series G 공시(2026-02) · 컴퓨트 연 ~3×는 프론티어 랩 공급 기준(Epoch AI) · 마진 38→70%+는 SemiAnalysis 추정 · 추론 비중 25→50%+는 Epoch AI.

HyperAccel

Bertha

가성비와 전성비를 높인 LLM 추론 전용 LPU

Process

Samsung 4nm · 500 mm²

Compute

FP16 384 TF · MXFP4 1.5 PF

Memory

LPDDR5X 192 GB · 546 GB/s · on-chip SRAM 256 MB

Power

TDP 250 W · PCIe Gen5

Serving

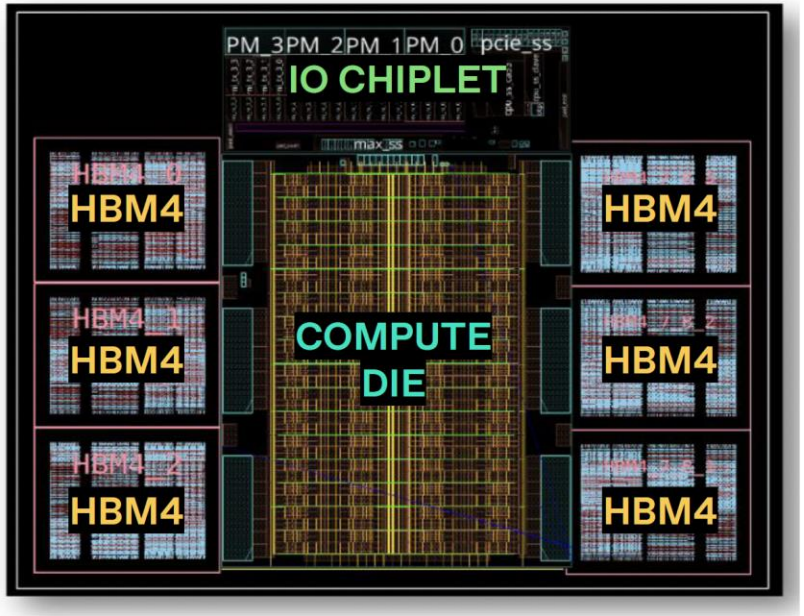
Batch 1–1024 · vLLM 플러그인

중요한 지표는 FLOPS가 아니라 tokens / $ 와 tokens / Watt · HBM 대신 LPDDR5X · 250 W · $5K

OpenAI

Specifications

Jalapeño ASIC	
Matrix compute	mxfp8 x mxfp8 = 3.4 PFLOP/s mxfp8 x mxfp4 = 6.7 PFLOP/s mxfp4 x mxfp4 = 13.4 PFLOP/s
Memory system	15.4 TB/s, 216 GiB
Scale-up network	Local = 128 ASICs @ 600 GB/s Global = 2048 ASICs @ 200 GB/s
Power	700 W
Jalapeño system (2048 ASICs)	
Matrix compute	mxfp4 x mxfp4 = 27 EFLOP/s
Memory system	32 PB/s, 432 TiB



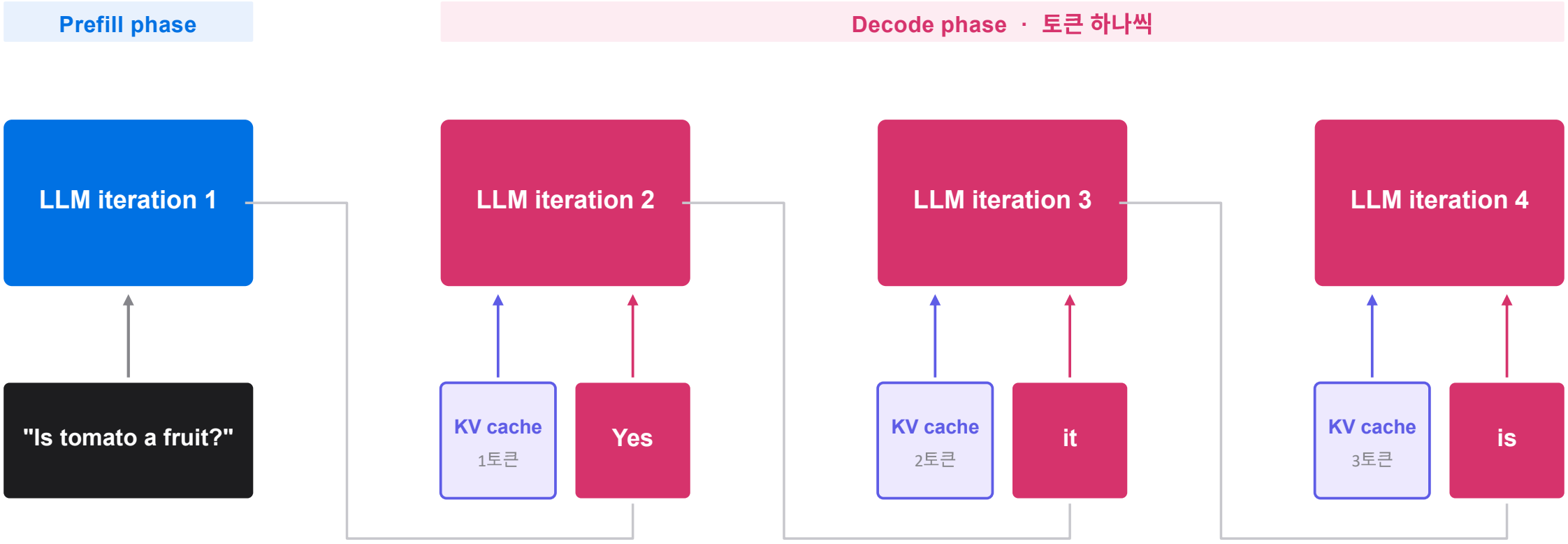
Package floorplan

Most important spec: perf/W on end-to-end workloads

LLM 추론의 구조

LLM은 두 단계로 돎 — Prefill과 Decode

한 번의 요청이 여러 번의 iteration으로 쪼개짐. 첫 iteration만 성격이 다름.



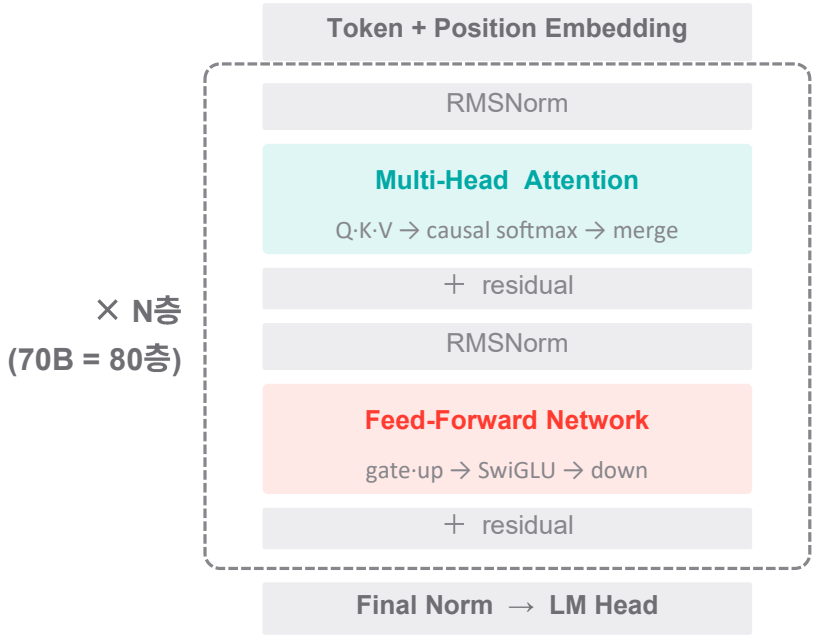
출력 토큰이 다음 iteration의 입력이 됨 — 그래서 순차적

그림 형식 출처 : Splitwise (Patel et al., ISCA 2024)의 prompt / token-generation phase 도식.

구조

Transformer — 핵심은 두 덩어리

현대 LLM은 디코더만 쌓음. 층 하나는 Attention과 FFN, 그리고 정규화와 잔차연결로 이뤄짐.



① Attention

층 파라미터의 18%

Q · K · V · O 네 개

② FFN

층 파라미터의 82%

gate · up · down 세 개

Multi-Head Attention

Q·K·V projection → causal softmax → output projection. 네 개의 가중치 행렬로 이뤄짐.

Feed-Forward Network

gate·up projection → SwiGLU → down projection. 세 개의 가중치 행렬이고 각각이 attention의 것보다 훨씬 큼.

그 밖

RMSNorm · residual · RoPE · LM head. 파라미터도 연산도 작음.

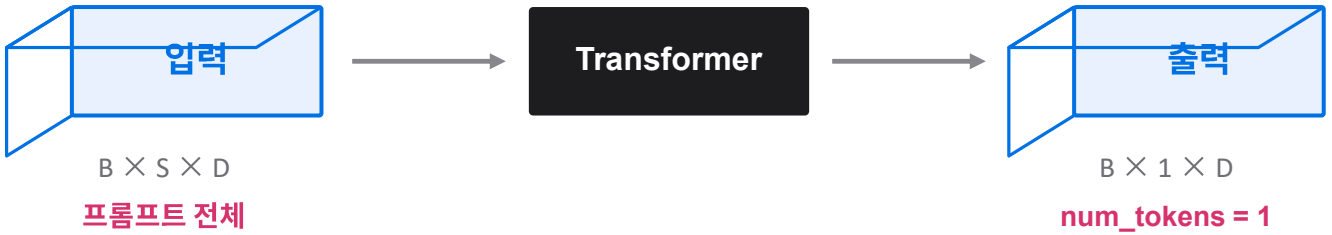
층 하나는 Attention 블록과 FFN 블록의 반복 · 70B는 이 층을 80번 쌓음 · 파라미터의 약 80%가 FFN에 있음(▲ 56.4 B / 70.6 B)

차원

입력과 출력은 3차원 — 그런데 하나가 1

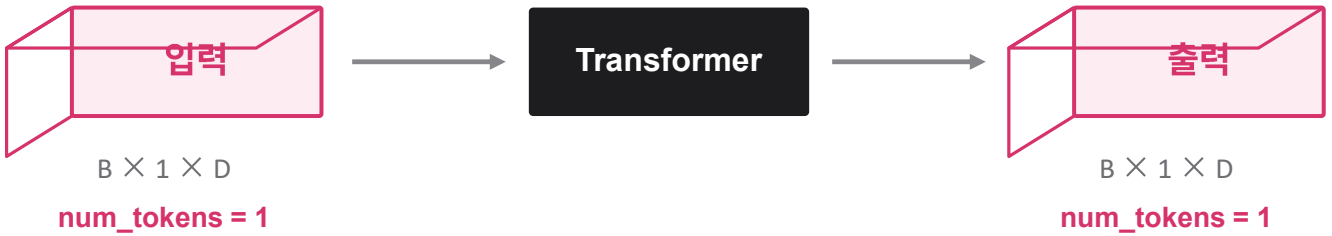
(batch, num_tokens, token_dim). Prefill의 출력과 Decode의 입·출력은 num_tokens = 1.

Prefill



프롬프트 S개를 한 번에 밀어 넣고
마지막 토큰 하나만 받음

Decode



토큰 하나를 넣고 하나를 받음
같은 가중치, 같은 커널

decode는 num_tokens가 1 — 뒤에 나올 「decode는 왜 memory-bound인가」가 전부 이 1에서 나온다

D = token_dim (모델 차원, Llama 3.1 70B는 8192) · S = 프롬프트 길이 · B = 배치 크기.

구조

Attention과 FFN의 연산 구성

두 덩어리 모두 결국 행렬곱. 차이는 「무엇과 곱하느냐」에 있음.

Attention

① Q · K · V generation

x @ Wq, Wk, Wv → 세 개의 텐서

② Multi-head self-attention

softmax($QK^T/\sqrt{d}$) · V · causal mask

③ Head merge

concat(heads) @ Wo → 다시 D 차원

토큰마다 다른 토큰을 봐야 함

FFN

① Gate · Up projection

x @ Wg, x @ Wu (D → D_ff, 보통 3.5~4배)

② SwiGLU

SiLU(gate) ⊙ up · 비선형

③ Down projection

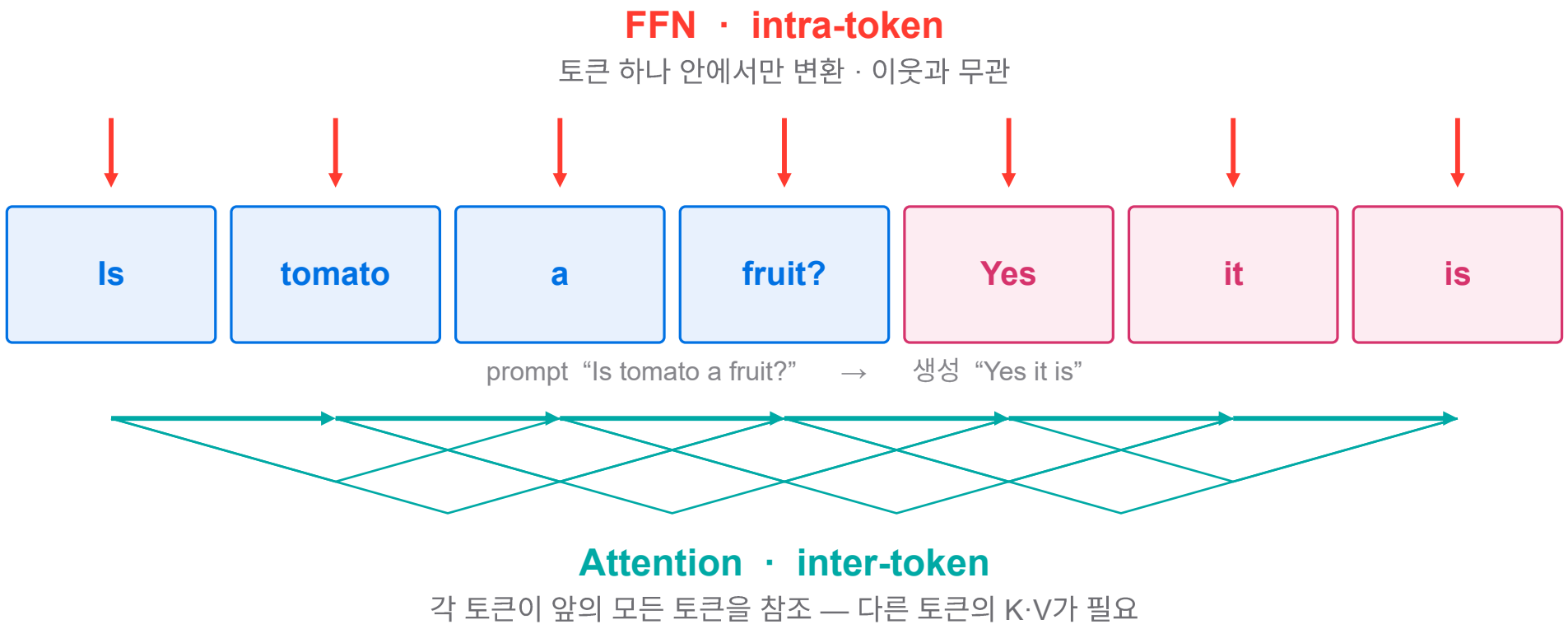
@ Wd → 다시 D 차원

토큰 하나만 보면 계산이 끝남

핵심 구분

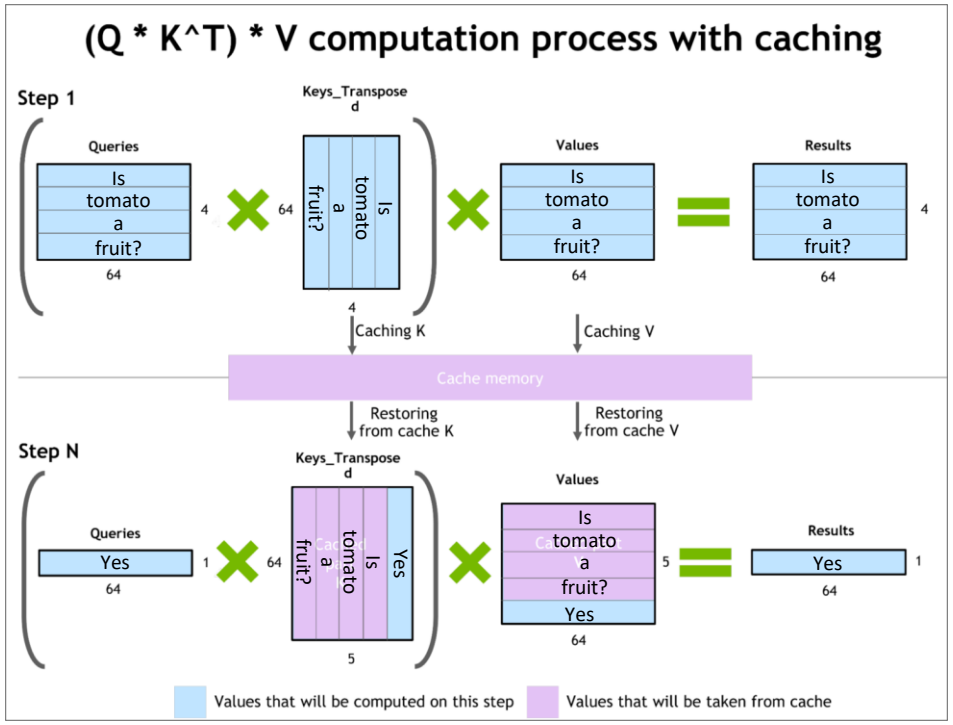
Attention은 토큰 사이, FFN은 토큰 안

inter-token vs intra-token. 이 구분이 배칭·KV·병렬화의 모든 차이를 만듦.



KV cache가 필요한 이유

스텝마다 앞 토큰의 K·V가 다시 필요 — 캐시가 없으면 매번 재계산



Step 1 · prefill

프롬프트 4토큰의 K·V를 계산 → 그대로 캐시에 적재

Step N · decode

새 토큰 1개의 K·V만 계산 · 나머지 4개는 캐시에서 복원

보라색 = 캐시에서

연산은 하늘색 한 줄뿐 · 나머지는 읽기만

그래서

KV cache는 최적화가 아니라 사실상 필수 · 대신 메모리를 먹음

Q는 매 스텝 새로 만드는 1행 · 누적되는 것은 K와 V뿐 · 캐시가 없으면 스텝 N에서 K·V를 N개 전부 다시 계산해 전체가 $O(S^2)$가 된다

KV cache

토큰 하나가 남기는 KV는 얼마인가

곱셈 다섯 개면 끝남 · Llama 3.1 70B 대입

$$\text{KV/token} = 2 \times L \times n\_kv \times d\_head \times \text{precision}$$

2
K와 V 두 벌

L
레이어 수 · 70B는 80층

n_kv
KV 헤드 수 · GQA로 64→8

d_head × p
헤드 차원 128 · BF16 2바이트

Llama 3.1 70B : $2 \times 80 \times 8 \times 128 \times 2 = 327{,}680$ B = 정확히 320 KiB / token

컨텍스트 128K를 채우면 한 세션이 320 KiB × 131,072 = 40 GB. H100 한 장(80GB)의 절반.

GQA가 없던 MHA(n_kv = 64)라면 8배인 2.5 MiB / token · GQA는 용량 대책이면서 동시에 대역폭 대책이다

모델 비교

모델마다 토큰 하나의 무게가 다름 — 52배까지

같은 128K 컨텍스트인데 KV cache가 67 GB에서 1.3 GB까지 벌어짐. 어텐션 구조 하나가 만드는 차이.

					컨텍스트별 KV cache 용량 · GB			
모델	구조	어텐션	층	KV / token	4K	32K	128K	1M
Llama 3.1 405B	Dense	GQA 8	126	504 KiB	2.11	16.91	67.6	541
Llama 3.3 70B	Dense	GQA 8	80	320 KiB	1.34	10.74	42.9	344
MiniMax-M2	MoE	GQA 8	62	248 KiB	1.04	8.32	33.3	266
Qwen3-235B-A22B	MoE	GQA 4	94	188 KiB	0.79	6.31	25.2	202
GLM-5.3	MoE	MLA + DSA	78	88 KiB	0.37	2.94	11.8	94
gpt-oss-120b	MoE	GQA 8 + SWA	36	72 KiB	0.30	2.42	9.7	77
DeepSeek-V3	MoE	MLA	61	69 KiB	0.29	2.30	9.2	74
Kimi K3	MoE	KDA 69 + MLA 24	93	27 KiB	0.11	0.91	3.6	29
DeepSeek-V4-Pro	**MoE**	**CSA + HCA**	**61**	**10 KiB**	**0.04**	**0.32**	**1.3**	**10**

⚠ gpt-oss-120b는 36층 중 18층이 sliding window, DeepSeek-V4-Pro는 전 층이 압축 + 윈도우 — 두 모델의 표 값은 상한이고 실제 상주량은 더 작다

BF16 KV · 시퀀스 1개 · 전 층 full-context 가정 · 아키텍처 값은 HuggingFace config.json 실물 확인(2026-08).

서빙 규모

필요 GPU 장수 (최소)

이 모델을 돌리려면 GPU가 몇 장 필요한가

batch 32 · 컨텍스트 128K · 출시 정밀도 기준. 저장 공간만 따진 하한 — 속도는 따로 봐야 함.

모델	정밀도	가중치 (GB)	KV (GB)	합계 (GB)	H100 80 GB	H200 141 GB	B200 180 GB	B300 · GB300 288 GB	Vera Rubin 288 GB
Llama 3.1 405B	BF16	812	1,082	1,894	27	15	12	8	8
Llama 3.3 70B	BF16	141	687	828	12	7	6	4	4
MiniMax-M2	FP8	229	533	761	11	6	5	3	3
Qwen3-235B-A22B	BF16	470	404	874	13	7	6	4	4
GLM-5.3	FP8	753	188	942	14	8	6	4	4
gpt-oss-120b	MXFP4	66	155	220	4	2	2	1	1
DeepSeek-V3	FP8	671	147	818	12	7	6	4	4
Kimi K3	INT4	1,400	58	1,458	21	12	9	6	6
DeepSeek-V4-Pro	**FP8**	**1,700**	**21**	**1,721**	**24**	**14**	**11**	**7**	**7**

맨 아랫줄 — DeepSeek-V4-Pro는 KV가 21 GB로 가장 작은데 필요한 장수는 가장 많다 · KV를 극단적으로 누르면 제약이 가중치로 되돌아온다

▲ 파생값 · 가중치는 각 모델의 배포 정밀도, KV는 FP8, 가용 메모리는 용량의 90% 가정 · Vera Rubin은 발표 사양(288 GB HBM4) 기준이며 아직 출하 전.

용량 대책

용량 문제를 푸는 네 갈래

파라미터를 줄이거나, 비트를 줄이거나, KV 구조를 바꾸거나, 아예 GPU 밖으로 보내거나.

① 파라미터

작은 모델

파라미터 자체를 줄임. MoE는 총 파라미터는 크지만 토큰당 활성만 계산함 — 다만 저장은 총 파라미터만큼 필요하다는 함정이 있음.

② 정밀도

양자화

FP8 · MXFP4 · NVFP4로 가중치 바이트를 절반·4분의 1로. 연산 경로까지 내리면 문턱은 그대로고 속도만 오름.

③ 아키텍처

KV 구조 변경

GQA로 KV 헤드를 줄이고, MLA로 잠재벡터 하나에 압축하고, SWA로 오래된 토큰을 잊음. KV 양자화도 여기.

④ 계층화

계층적 KV

지금 안 쓰는 세션의 KV를 CPU 메모리나 SSD로 내려두고 필요할 때 다시 올림. 용량은 벌지만 재로드 지연이 TTFT로 돌아옴.

성능의 물리학

xPU 성능은 두 가지가 정함

연산과 I/O. 둘 중 느린 쪽이 커널의 시간을 정함 — 빠른 쪽은 그동안 대기.

① 연산 Compute

무엇	**곱셈-덧셈을 초당 몇 번**
단위	**FLOP/s**
H100 BF16 dense	**989.5 TFLOPS**

텐서코어가 담당함. 데이터시트의 큰 숫자는 대개 sparsity 포함이라 dense는 그 절반.

연산이 느리면 → compute-bound

② I/O Memory

무엇	**메모리에서 초당 몇 바이트**
단위	**Byte/s**
H100 HBM3	**3.35 TB/s**

HBM이 담당함. 세대가 바뀌어도 연산만큼 빨리 늘지 않음 — 이게 메모리 벽.

메모리가 느리면 → memory-bound

둘 다 peak 스펙 · 실측은 언제나 이보다 낮다

Ridge point — 둘이 균형을 이루는 한 점

어느 쪽도 병목이 아닌 지점. 칩 하나를 한 숫자로 요약함.

Ridge point = Peak FLOPS ÷ Memory Bandwidth

H100 : 989.5 TFLOP/s ÷ 3.35 TB/s = 295 FLOP/Byte

295

바이트 하나를 가져오는 동안
계산을 295번 해야 본전

AI < 295

내 커널이 memory-bound
텐서코어가 유휴

AI > 295

내 커널이 compute-bound
HBM이 유휴

ridge가 클수록

칩이 요구하는 AI가 높음
최대 성능 뽑기가 어려움

정의 출처 : Williams et al., "Roofline", CACM 2009 — "the minimum operational intensity required to achieve maximum performance".

Arithmetic Intensity — 바이트 하나로 몇 번 계산하나

칩이 아니라 커널의 성질 · 같은 칩에서도 커널마다 다름

Yes

x_{11}	x_{12}	x_{13}	x_{14}

X

w_{11}	w_{21}	w_{31}	w_{41}
w_{12}	w_{22}	w_{32}	w_{42}
w_{13}	w_{23}	w_{33}	w_{43}
w_{14}	w_{24}	w_{34}	w_{44}

=

y_{11}	y_{12}	y_{13}	y_{14}
y_{21}	y_{22}	y_{23}	y_{24}
y_{31}	y_{32}	y_{33}	y_{34}
y_{41}	y_{42}	y_{43}	y_{44}

가중치 W는 모든 행이 나눠 쓰고, 입력 X와 출력 Y만 행 수 M에 비례해 늘어남

행렬 × 행렬

2 n^3 FLOP

AI ∝ n

행렬 × 벡터

2 n^2 FLOP

AI = 2 / p

Prefill

M = 토큰 수천 개

AI 수백~수천

Decode

M = 배치 크기

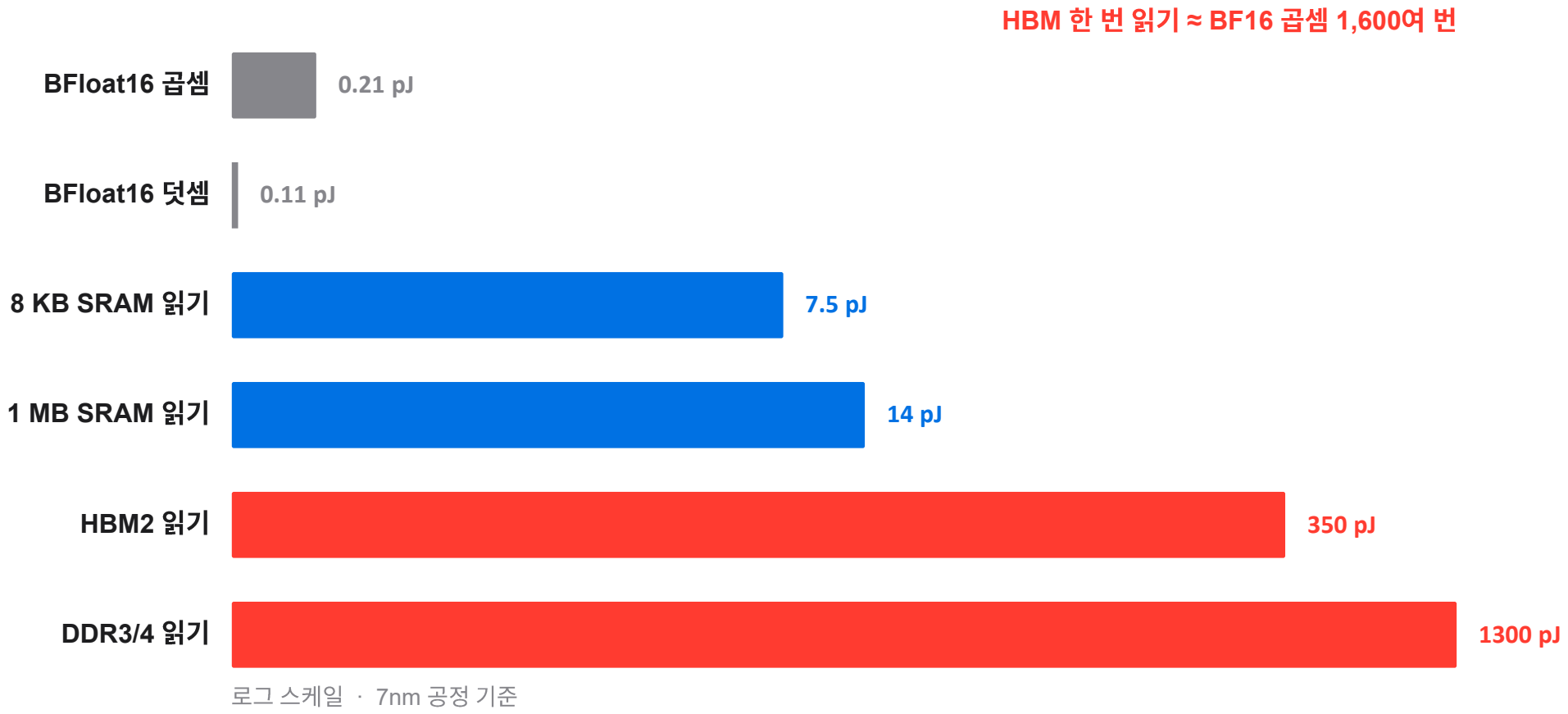
AI = 배치

정확한 GEMM 형태는 AI = 1 / (1/M + 1/K + 1/N) — M이 K · N보다 훨씬 작을 때만 AI ≈ M

Arithmetic Intensity

왜 AI가 중요한가 — 메모리 접근이 압도적으로 비쌈

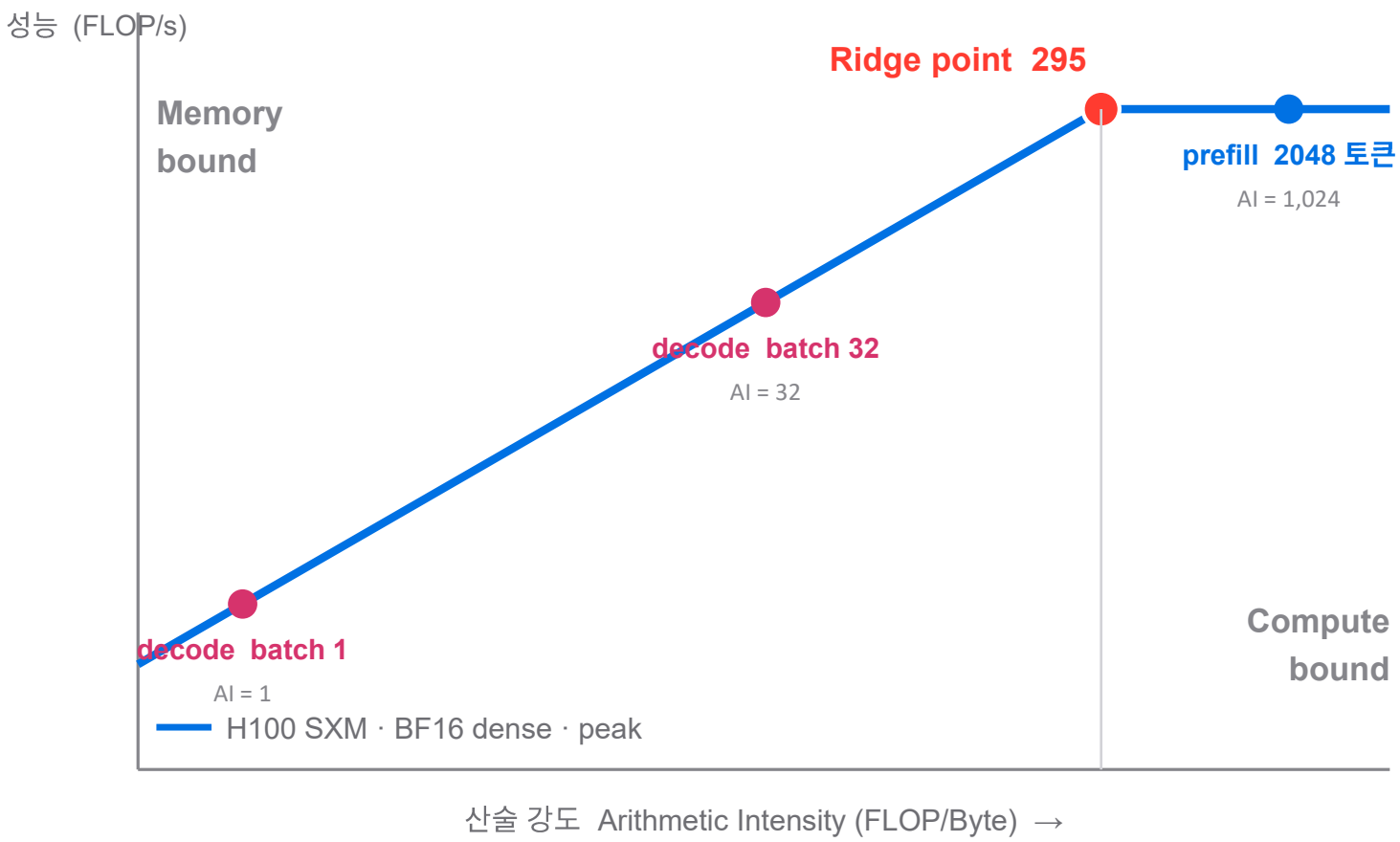
느려서만이 아님. 에너지가 자릿수로 차이 남. 같은 일을 하고도 전력이 몇 백 배 듦.



출처: Horowitz "Computing's Energy Problem" 계열 표를 7nm으로 스케일한 값(참조 자료 10쪽). DRAM 수치는 접근 패턴에 따라 크게 흔들리므로 자릿수 감각으로만 쓸 것.

Roofline — 두 개의 천장

사선은 대역폭이 만드는 천장, 평지붕은 연산이 만드는 천장. 둘이 만나는 곳이 ridge point.



사선 구간

성능 = 대역폭 × AI. 로그-로그에서 기울기가 정확히 1.

평지붕 구간

성능 = peak FLOPS. AI를 더 올려도 안 오름.

decode는 왼쪽 끝

배치 32면 AI 32 — 연산 능력의 32/295, 약 11%만 씀.

prefill은 오른쪽

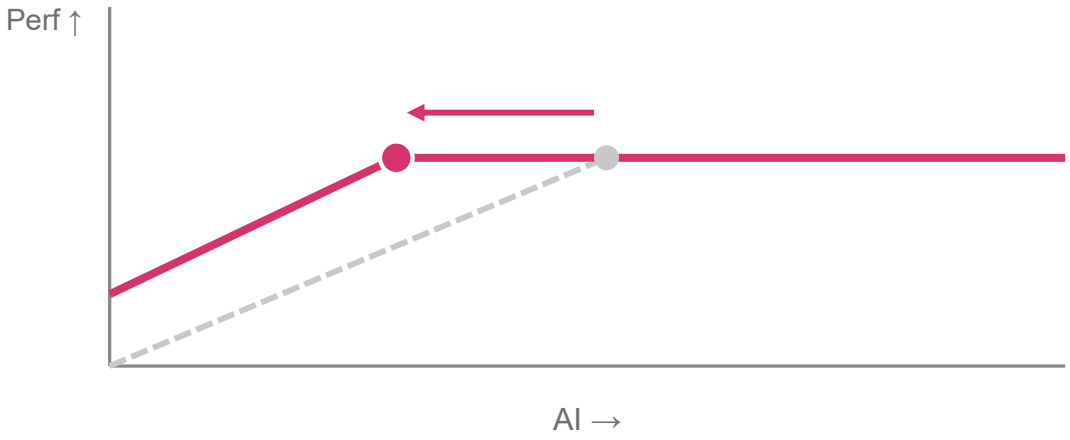
토큰 수천 개를 한 번에 밀어 AI가 수백~수천이 됨.

Roofline

하드웨어가 좋아지면 천장은 어떻게 변하나

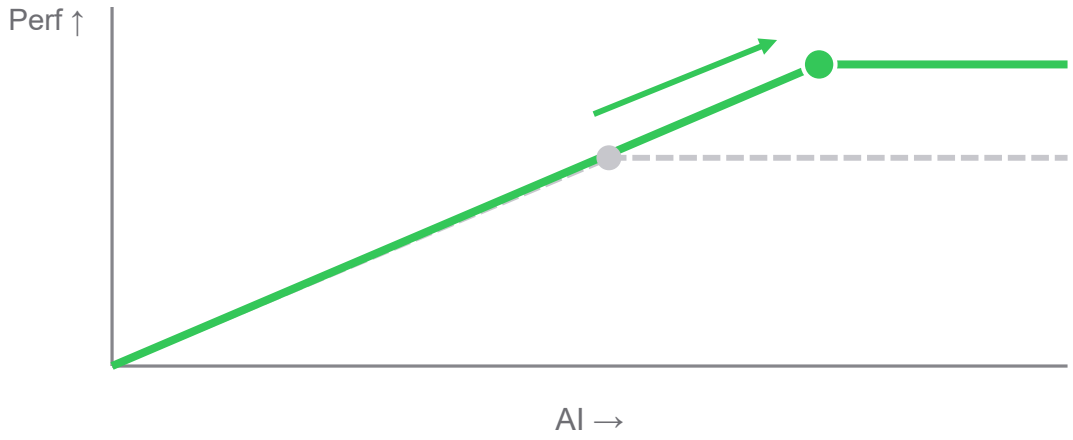
대역폭은 사선을 세우고, 연산은 지붕을 들어 올림. 두 개선은 방향이 다름.

① Bandwidth ↑



사선이 섬 → ridge가 왼쪽으로
memory-bound 커널이 빨라짐

② Compute ↑



천장이 오름 → ridge가 오른쪽으로
compute-bound 커널이 빨라짐

before　after · 대역폭 ↑　after · 연산 ↑

그래서 「이 칩이 더 좋다」는 말은 내 커널의 AI가 어디인지를 밝히지 않으면 성립하지 않는다 · decode처럼 왼쪽에 있으면 대역폭 개선만 체감된다

Decode에서 AI를 올리는 법 — 배치

행렬-벡터를 행렬-행렬로 바꿈. 가중치는 한 번만 읽고 여러 요청이 나눠 씀.

batch 1

GEMV

가중치 바이트는 똑같이 한 번

행 하나를 위해 W 전체를 읽음.
바이트당 계산 1회 → 텐서코어는 유휴

AI = 1

batch 4

GEMM

가중치 바이트는 그대로, 계산만 4배

같은 W를 읽어 4행을 처리함.
바이트당 계산 4회 → AI가 배치 크기

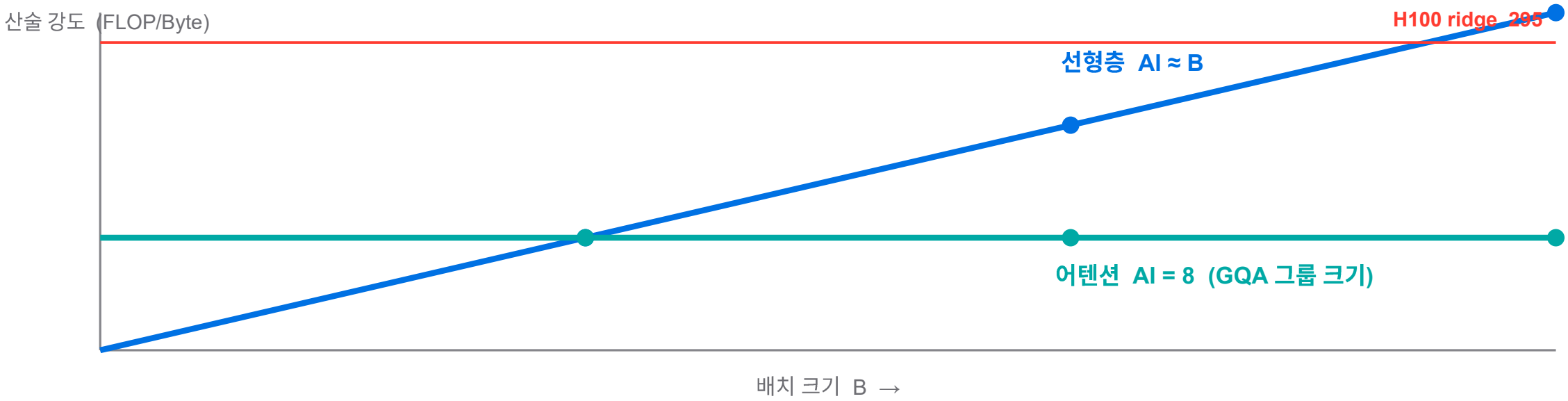
AI = 4

roofline은 계단이 아니라 경사선이라, ridge point까지 못 가도 AI를 올린 만큼 빨라진다 · ⚠ 대신 배치를 키우면 TPOT(체감 속도)이 나빠지고, 보통 KV 용량이 먼저 막는다

어텐션의 예외

Attention은 배치를 키워도 AI가 안 오름

요청마다 자기 KV를 가짐. 배치가 2배면 연산도 2배, 읽을 바이트도 2배 — 그대로 약분됨.



요청마다 KV가 다름 → 배치를 2배 하면 연산도 2배, 읽을 바이트도 2배. 약분됨.

어텐션 decode의 AI = 2 × n_head / (n_kv × precision) · Llama 3.1 70B BF16이면 2 × 64 / (8 × 2) = 8, MHA였다면 1 — 배치도 시퀀스 길이도 식에서 사라진다

어텐션 대책

Attention에서 AI를 올리는 세 갈래

배치가 안 통하니 다른 손잡이가 필요함. 셋 다 「KV 바이트 하나를 몇 번 쓰느냐」를 건드림.

① 구조

헤드끼리 공유
GQA · MQA · MLA

KV 헤드 하나를 여러 Q 헤드가 나눠 씀. 공유 폭이 곧 AI — MHA 1, Llama 3.1 70B 8, MQA는 헤드 수 전체.

MLA는 한 걸음 더 나감. 모든 헤드가 압축된 잠재벡터 하나를 공유하고, 흡수(absorption) 형태에서는 헤드당 내적 차원까지 커짐.

② 재사용

배치끼리 공유
Prefix caching

서로 다른 요청이 같은 접두사를 쓰면 그 구간 KV를 공유함. 시스템 프롬프트·few-shot·에이전트의 반복 컨텍스트에서 특히 큼.

읽는 바이트가 줄어드는 게 아니라 「안 읽어도 되는 구간」이 생기는 것이라, AI가 아니라 총량을 줄이는 쪽에 가까움.

③ 정밀도

바이트 자체를 줄이기
KV 양자화

KV를 FP8·INT4로 저장함. 분모가 그만큼 줄어드니 AI는 바로 오름.

용량도 같이 벌어서 배치를 더 키울 수 있음. prefix caching과 충돌하지 않음 — 블록 해시에 dtype이 안 들어가기 때문.

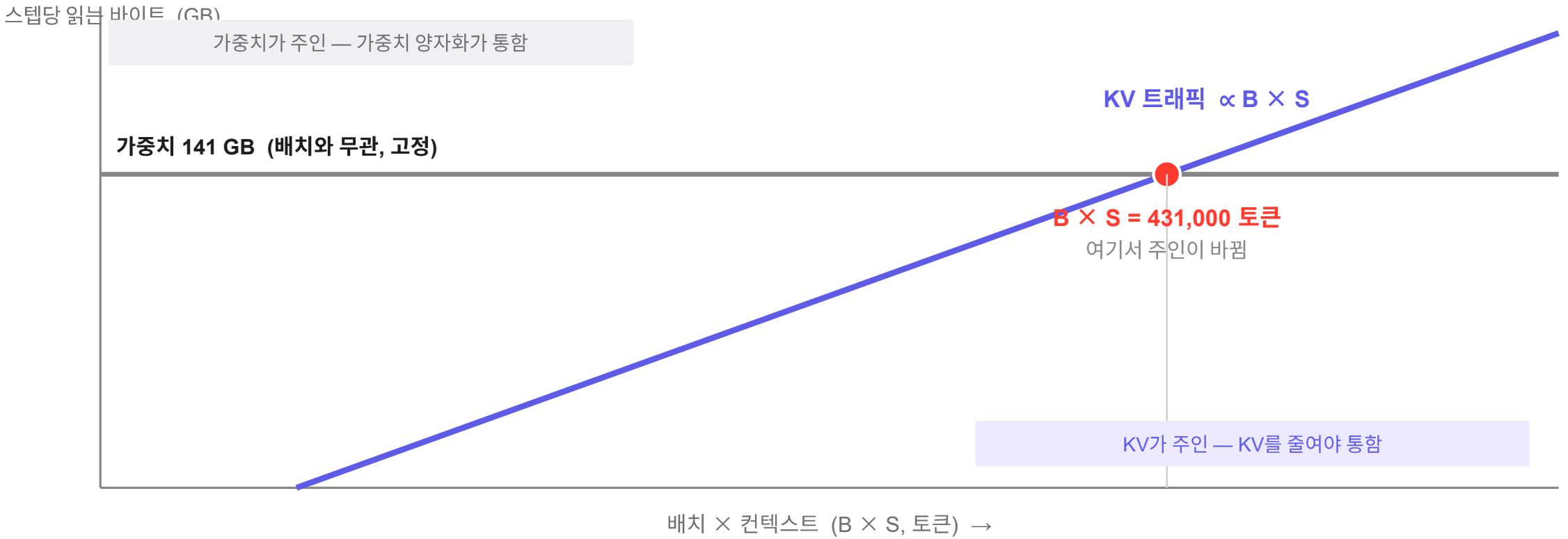
대가는 정확도. 특히 긴 컨텍스트에서 민감함.

셋은 겹쳐 쓴다 · DeepSeek은 MLA + FP8 KV, 상용 서빙은 대부분 GQA + prefix caching + FP8 KV 조합

트래픽의 주인

가중치가 중요한가, KV가 중요한가

배치 × 컨텍스트가 43만 토큰을 넘으면 주인이 바뀜. 그때부터 가중치 양자화는 통하지 않음.



▲ 파생값 · Llama 3.1 70B BF16 기준 : 가중치 141.2 GB ÷ 토큰당 KV 320 KiB = 430,908 토큰 (활성값·워크스페이스 제외).

트래픽의 주인

배치를 키우면 컨텍스트 얼마에서 KV가 가중치를 넘나

Llama 3.1 70B · 가중치 141.2 GB 고정 · 토큰당 KV 320 KiB. 배치와 컨텍스트의 곱이 43만을 넘으면 주인이 바뀜.

	KV = 가중치가 되는 지점		컨텍스트 128K일 때	
배치 B	교차 컨텍스트 S*		128K에서 KV	가중치 대비
1	430,908 토큰	421K	43 GB	0.3×
4	107,727 토큰	105K	172 GB	1.2×
8	53,864 토큰	53K	344 GB	2.4×
16	26,932 토큰	26K	687 GB	4.9×
32	**13,466 토큰**	**13K**	**1,374 GB**	**9.7×**
64	**6,733 토큰**	**7K**	**2,749 GB**	**19.5×**
128	**3,366 토큰**	**3K**	**5,498 GB**	**38.9×**
256	**1,683 토큰**	**2K**	**10,995 GB**	**77.9×**
512	**842 토큰**	**1K**	**21,990 GB**	**155.7×**

B × S* = 431,000 토큰으로 항상 일정 — 배치를 2배 키우면 교차 컨텍스트는 정확히 절반이 된다 · 에이전트 워크로드의 입력 중앙값 14만 토큰이면 배치 4부터 KV가 가중치보다 무겁다

▲ 파생값 · S* = 141.2 GB ÷ (B × 320 KiB) · 활성값·워크스페이스 제외.

InferenceX

「초당 몇 토큰」이라는 한 숫자로는 아무것도 알 수 없음

똑같이 초당 1만 토큰이라도 의미가 정반대. 그래서 추론 성능은 반드시 두 축으로 봄.

한 명에게 빠르게

가로축	**tok/s/user**
이름	**Interactivity**
배치	**작음**

사용자는 만족함. 하지만 GPU가 놀아서 전체 처리량이 적고, 토큰당 비용이 비쌈.

= 높은 Interactivity

여러 명에게 나눠서

세로축	**tok/s/GPU**
이름	**Throughput**
배치	**큼**

장비를 알차게 씀. 대신 각 사용자는 답을 더 기다림 — TPOT이 나빠짐.

= 높은 Throughput

출처 : InferenceX (SemiAnalysis) — 이 두 축 위에 여러 엔진·하드웨어를 올려 비교하는 공개 벤치마크.

핵심 그래프 — 먼저 두 축

가로는 한 사람이 받는 속도, 세로는 GPU 한 장의 처리량. 이 평면 위에서만 성능을 말할 수 있음.

가로축

한 사람이 초당 몇 토큰 받나 — interactivity

세로축

GPU 한 장이 초당 몇 토큰 — throughput

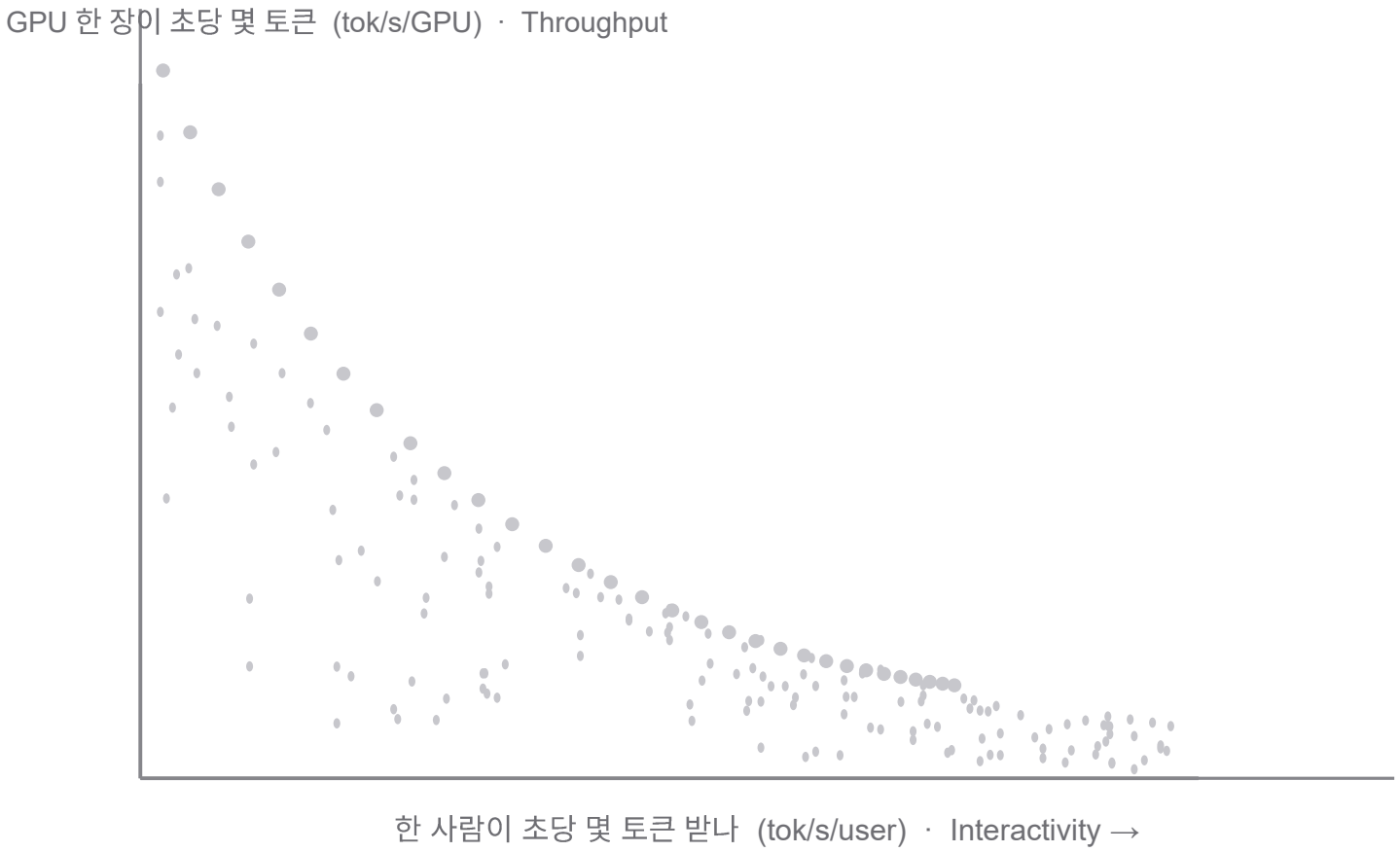
한 점 = 한 실험

동시성·병렬화·프레임워크·정밀도를 하나씩 고정하고 돌린 결과

InferenceX

핵심 그래프 — 점을 찍으면

설정을 바꿔가며 돌린 결과가 평면에 흩어짐. 아직 곡선은 없음.



점 하나 = 설정 하나

동시성 4·8·16·32·64… 를 스윕한 결과

안쪽 점

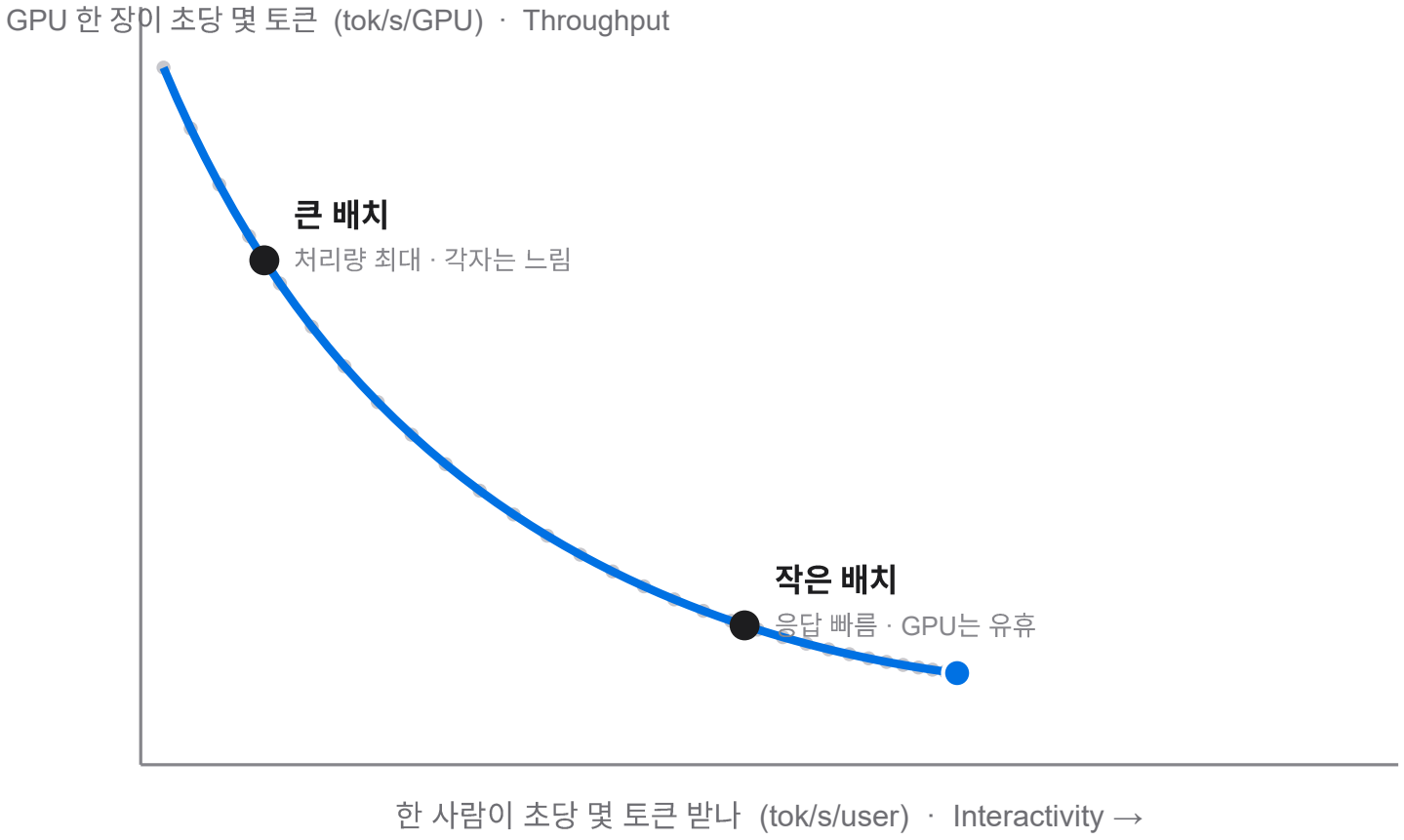
더 좋은 설정에 지배당한 점 — 쓸 이유가 없음

바깥 점

어느 쪽으로도 더 나아갈 수 없는 점

핵심 그래프 — 파레토 프론티어

바깥 점들을 이으면 경계가 나옴. 한쪽을 얻으면 다른 쪽을 내줌.



곡선 위로는 못 감

달성 가능한 최선의 경계

오른쪽 = 작은 배치

토큰마다 가중치 전체를 다시 읽음 → 대역폭 병목

왼쪽 = 큰 배치

한 번 읽어 많이 계산함 → 연산 병목

⚠ 곡선은 볼록함

원점 쪽으로 볼록(convex) · 오목하게 그린 그림이 많은데 실측과 반대

▲ 측정된 파레토 프론티어의 정규화 면적 0.313~0.323 · 형상 적합 t = 1.767 ms + 0.474·B^0.600.

Token Throughput per GPU vs. Interactivity

gpt-oss 120B • FP4 • 8K / 1K • Source: SemiAnalysis InferenceX™ • Updated: 04/14/2026

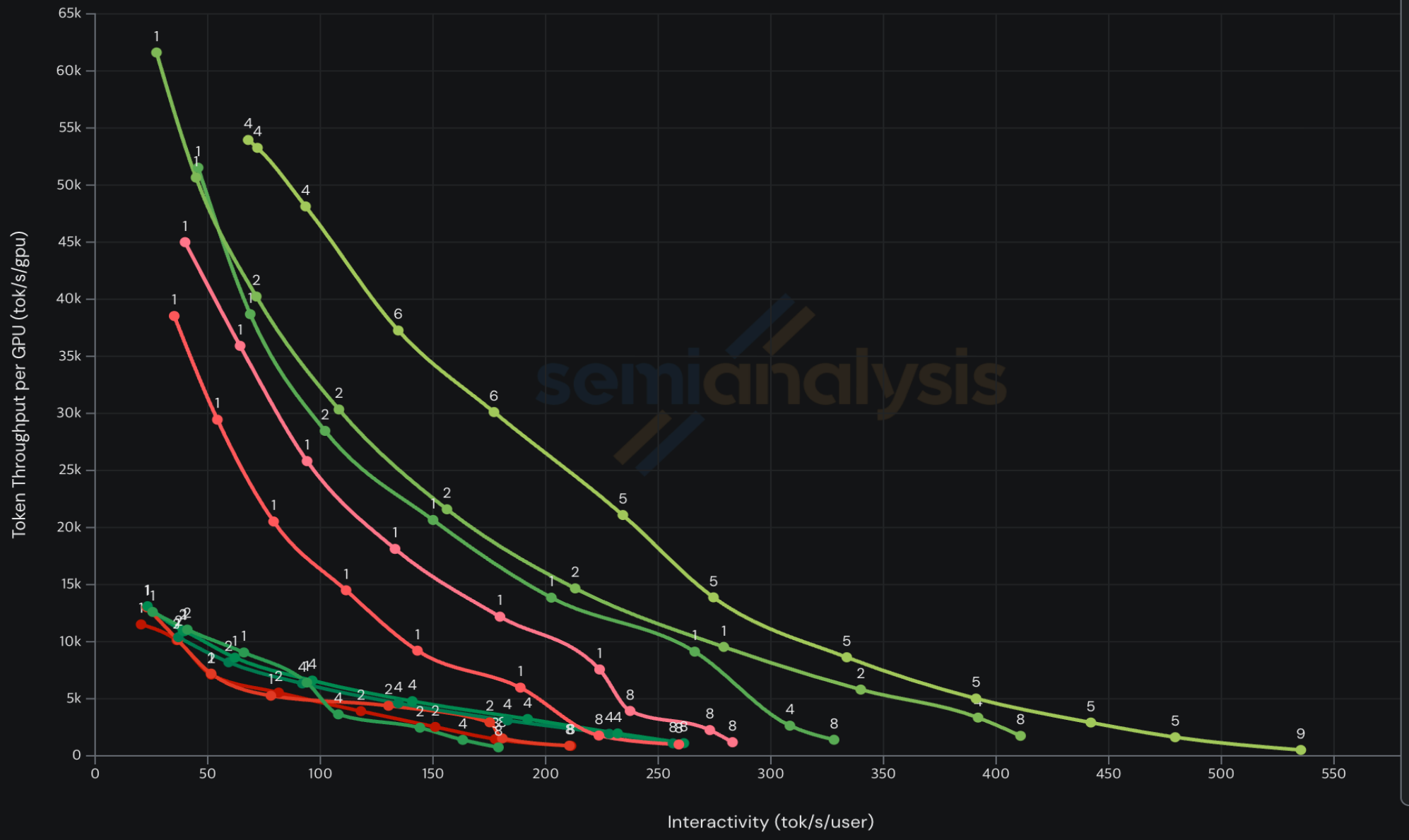


Chart | Table

Search...

- GB200 NVL72 (Dynamo TRT)
- B200 (TRT)
- B200 (vLLM)
- MI355X (ATOM[1])
- MI355X (vLLM)
- H200 (TRT)
- H200 (vLLM)
- MI325X (vLLM)
- H100 (vLLM)
- MI300X (vLLM)

Log Scale

Optimal Only

Hide Labels

High Contrast

Parallelism Labels

Gradient Labels

Line Labels

Collapse

[1] The ATOM engine is promising, however it has yet to serve production tokens. It is still in its infant stage.

Shift+Scroll to zoom • Drag to pan • Double-click to reset • Click a point to pin tooltip

올바른 사용법 — 세로로 비교하라

한 점(single metric)만 보면 반드시 오해함. 순서를 지켜야 함.

1

목표 응답 속도부터 정하기

챗봇은 사람이 읽는 속도인 초당 10~20토큰이면 충분함. 추론·코딩 에이전트는 훨씬 빠른 속도를 요구함.

2

그 지점에 세로선 긋기

가로축에서 목표 interactivity에 해당하는 위치. 이 선을 안 그으면 비교 자체가 성립하지 않음.

3

세로선 위에서 곡선 높이 비교하기

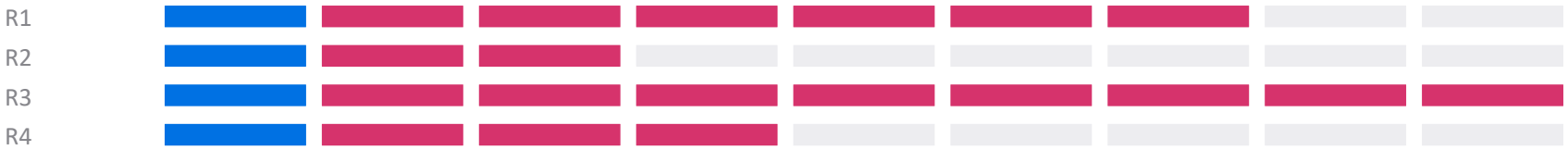
곡선이 높을수록 같은 체감 속도를 더 싸게 냄. 곡선이 교차하면 「어느 칩이 빠른가」의 답이 구간마다 뒤집힘.

곡선이 교차한다는 것이 핵심 — 「A가 B보다 몇 배 빠르다」는 문장은 가로축 좌표를 밝히지 않으면 참도 거짓도 아니다

① 요청 단위에서 iteration 단위로

가장 큰 개선은 커널이 아니라 배치를 언제 바꾸느냐였음.

요청 단위 (static batching)



가장 긴 요청이
끝날 때까지
전부 대기

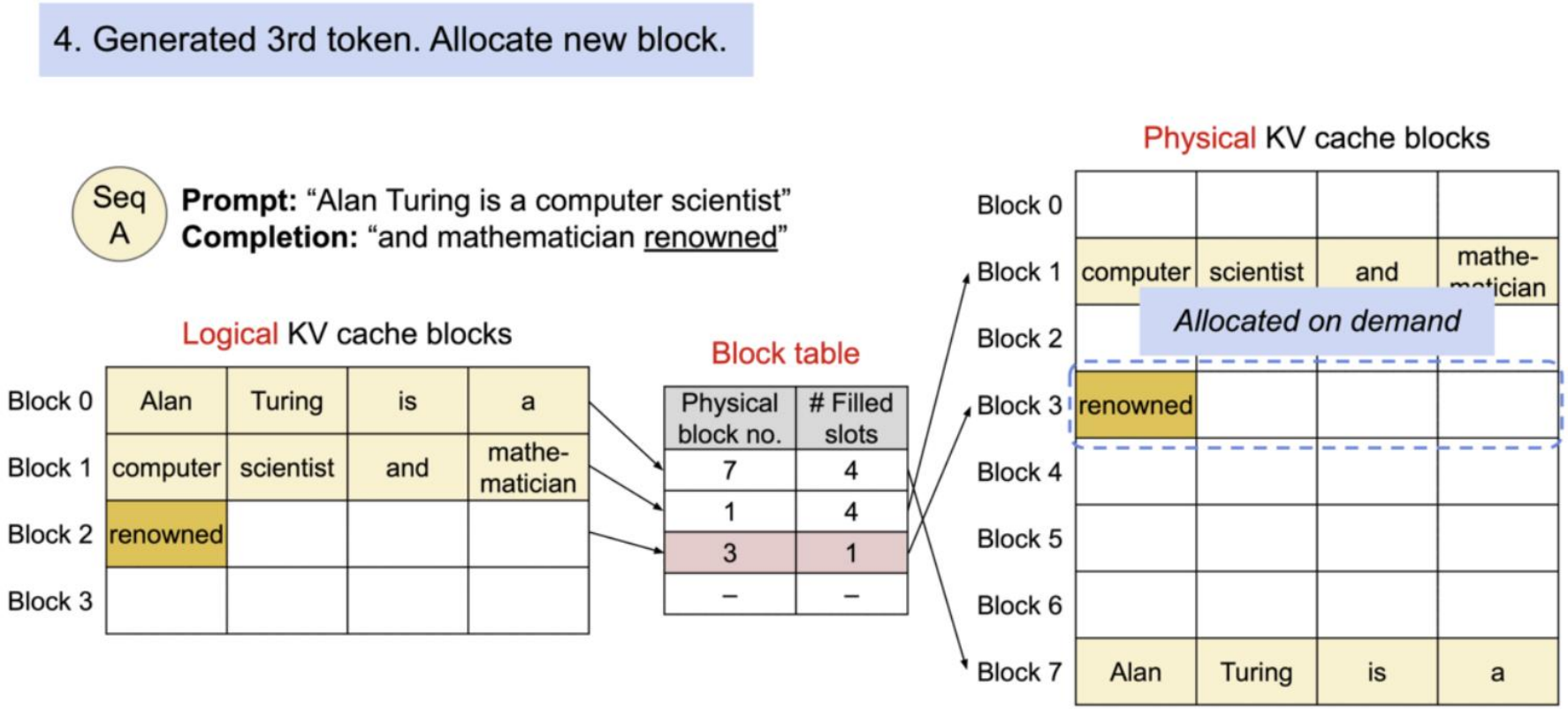
Iteration 단위 (continuous batching)

요청 단위 배칭은 가장 긴 요청이 끝날 때까지 나머지 슬롯이 전부 논다 · 길이가 제각각인 실제 트래픽에서 손실이 가장 크다

출처 : Orca (OSDI '22) — iteration-level scheduling. vLLM이 오픈소스로 보급.

② PagedAttention — KV를 페이지로 관리함

OS의 가상 메모리를 GPU로 가져왔음.



Example generation process for a request with PagedAttention.

③ Prefix caching — 요청끼리 KV를 나눠 씀

같은 접두사면 같은 KV. 블록 해시로 찾아서 계산을 통째로 건너뜀.

세 덩어리가 두 요청에
똑같이 들어 있음.

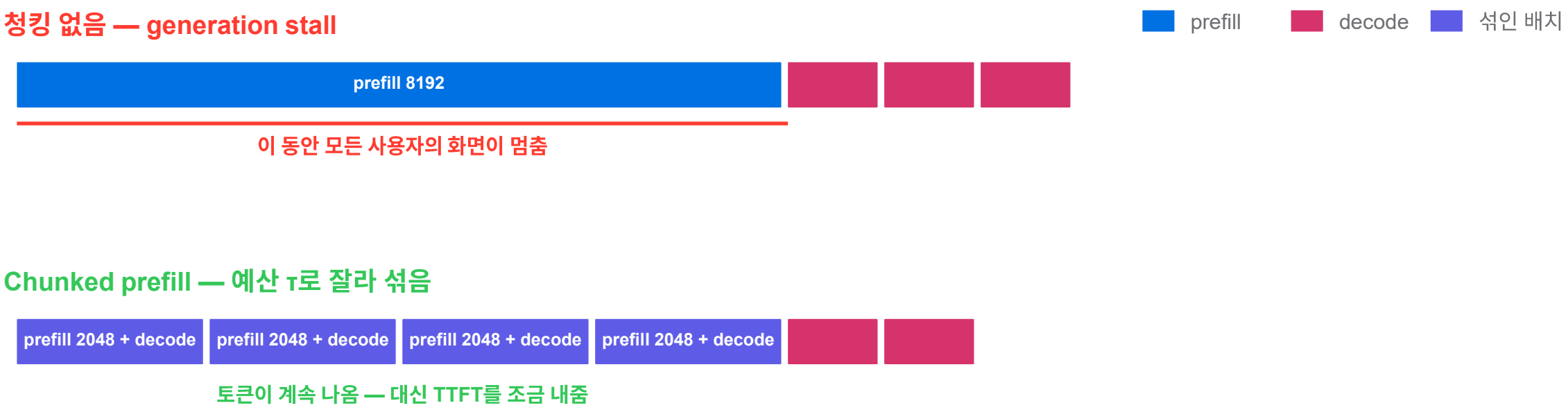
한 번만 계산하고
나머지는 재사용함.

에이전트는 매 턴 같은 접두사를 다시 보냄 — prefix caching이 있고 없고가 TTFT를 자릿수로 가름

블록 해시 = (부모 해시, 토큰 튜플, extra keys). extra keys에 KV의 dtype이 들어가지 않아 KV 양자화와 충돌하지 않음.

④ Chunked prefill — 긴 prefill을 잘라서 섞음

prefill 하나가 통째로 들어오면 그 iteration 동안 모든 decode가 멈춤. Sarathi가 generation stall이라 부른 것.



> **공짜가 아니라 거래 — TBT를 사고 TTFT를 판다 · chunked prefill 단독으로 P50 TTFT가 0.53초 → 1.04초 · 예산 τ를 512 이하로 내리면 prefill조차 memory-bound로 미끄러진다**

출처 : Sarathi-Serve (OSDI '24) — 청킹 없는 hybrid batching은 decode-only 대비 TBT를 최대 28.3배까지 밀어올림. Table 4 ablation.

⑤ Speculative decoding — 배치를 인위적으로 부풀림

작은 모델이 초안을 쓰고 큰 모델이 한 번에 검증함. 틀리면 버림 — 출력 분포는 그대로.

① 작은 초안 모델이 8개를 싸게 뽑음

② 큰 모델이 8개를 한 번에 검증함 — forward 한 번

핵심: GPU는 이 8개가 서로 다른 사용자 것인지 한 사용자의 투기 토큰인지 구분하지 못함

즉 투기 디코딩은 배치를 인위적으로 부풀리는 장치. 배치 8 + 초안 8개 = 배치 64와 똑같이 생긴 행렬곱.

그래서 memory-bound 구간에서 놀던 연산 유닛을 공짜로 채움 — 무릎 아래에서는 검증이 공짜.

배치를 키우면 사용자마다 KV가 따라 늘지만, 투기 디코딩은 사용자를 늘리지 않고 decode의 산술 강도만 올린다 — KV를 늘리지 않고 무릎에 다가가는 유일한 손잡이

▲ 345 ÷ 8 = 43 · EAGLE-3의 배치 64 · 1.38배는 H100 · SGLang · MT-Bench 조건.

다섯 가지가 각각 무엇을 건드리나

능선은 칩의 숫자. 소프트웨어는 「커널에 무엇을 넣을지」만 바꿈 — 그래서 다섯 개가 전부 같은 열에 모임.

기법	능선 (칩)	AI(M) 곡선 (모델)	M(B) 지도 (소프트웨어)	1차 효과
Iteration 스케줄링	—	—	빈 슬롯을 즉시 채움	처리량 · 대기시간
PagedAttention	—	—	배치를 키울 여지를 만듦	KV 실효율 20% → 96%
Prefix caching	—	—	계산할 M 자체를 줄임	TTFT
Chunked prefill	—	—	M = τ 로 고정함	TBT ↔ TTFT 거래
Speculative decoding	—	—	M = B(γ+1)	저배치 지연

능선 열이 전부 비어 있다는 것이 이 표의 전부 · 능선은 칩을 바꾸거나 칩의 연산 경로를 바꿀 때만 움직인다 · 「엔진을 바꿨더니 하드웨어가 좋아진 것 같다」는 착시가 여기서 나온다

Prefill과 Decode는 Roofline의 반대편에 있음

같은 모델, 같은 커널인데 하나는 연산이 병목이고 하나는 대역폭이 병목. 섞으면 둘 다 손해.

Prefill

한 번에 미는 토큰	**S · B개 (수백~수십만)**
연산량	**≈ 2 · P · S · B**
가중치 로드	**2P 바이트 한 번**
산술 강도	**≈ S · B**

DistServe 측정: 13B 모델에서 512토큰 prefill 하나가 이미 A100을 near compute-bound로 만듦.

compute-bound → 연산이 병목

Decode

한 번에 미는 토큰	**B개 (배치 크기)**
연산량	**≈ 2 · P · B**
가중치 로드	**똑같이 2P 바이트**
산술 강도	**≈ B**

B가 수백 이하면 HBM 대역폭에 묶임. 배치 32면 텐서코어 활용률 상한이 11%.

memory-bound → 대역폭이 병목

▲ A100 ridge 153 · OPT-13B 형상(K = N = 5120)에서 문턱 토큰 M* = 163 · 512토큰이면 문턱의 3.1배.

Disaggregation · 간섭

섞으면 간섭이 양방향으로 일어남

두 방향의 크기가 다름. 큰 쪽을 이름으로 부르지 않으면 대책을 잘못 세움.

① 큰 쪽

prefill → decode

decode 스텝의 시간을 그 배치에 낀 prefill 토큰 수가 지배함.
decode 64개에 prefill 2048이면 M의 97%가 prefill.

▲ 짧은 컨텍스트에서 5.9배 · 8192를 통째로 넣으면 22배.
이것이 generation stall.

② 작아 보이지만

decode → prefill

GPU가 포화면 decode가 prefill의 TTFT를 늘림.
「포화」가 두 가지.

연산 포화 — ▲ 최대 12%대.
토큰 예산 포화 — TTFT ∝ 1/(예산−B)로 터짐. ▲ 최대 16배.

진단

본질

TTFT와 TPOT가 「배치 크기」라는 노브 하나를 놓고 싸우는 두 개의 SLO.

colocated serving은 한 인스턴스가 두 지표를 동시에 최적화하도록 강요함.

그래서 둘 중 하나를 포기하거나 자원을 과다 투입함.

청크를 작게 잡을수록 decode가 prefill을 더 크게 지연시킨다(512에서 12.5%, 8192에서 0.8%) · 그런데 청크를 작게 잡는 이유가 바로 decode 보호

▲ 파생값 · 컨텍스트가 길어지면 배수는 오히려 줄어듦(128K에서 2.1배) — decode 자신의 KV 비용이 이미 커졌기 때문. 대신 절대 지연은 커짐.

Disaggregation은 무엇을 좋게 하나

「병렬로 돌려서 빨라진다」가 아님. 간섭이 사라진 decode 인스턴스가 배치를 훨씬 크게 키울 수 있기 때문.

requests →

→ tokens

Prefill 풀

compute-bound

TP·EP 크게 · 작은 배치

목표 = 빠른 TTFT

KV cache 전송

NIXL · RDMA / NVLink

Decode 풀

memory-bound

배치를 문턱까지 크게

목표 = 균일한 TPOT

① 간섭이 사라짐

decode 풀은 decode만 돎. iteration 시간이 균일해져서 스트림이 안 끊김.

② 배치를 키울 수 있음

TPOT SLO를 지키면서 문턱까지 밀어 올릴 수 있음 — 이게 1차 원인.

③ 따로 튜닝함

단계별로 batch·병렬화·GPU 비율을 독립적으로 최적화함.

총 연산량은 그대로다 — 그래서 throughput은 안 오른다 · 오르는 것은 SLO를 지킨 토큰의 비율, 즉 goodput이다

출처 : vLLM 문서 — "Disaggregated prefill DOES NOT improve throughput." · DistServe ablation — 모든 병렬화 조합을 탐색한 colocated "vLLM++"가 기본 vLLM과 성능이 같았음.

throughput은 왜 안 오르고 goodput만 오르나

같은 GPU가 같은 연산을 한다. 바뀌는 것은 「언제」뿐임.

throughput · 안 오름

세는 것 **초당 토큰 전부**

바뀌는가 **안 바뀜**

분리해도 곱해야 할 행렬은 그대로고, GPU 총량도 그대로임. 오히려 KV를 인스턴스 사이로 옮기는 비용이 붙어 조금 손해임.

연산 총량이 안 변하므로 안 오름

goodput · 오름

세는 것 **SLO를 지킨 토큰만**

바뀌는가 **크게 오름**

섞여 있을 때는 prefill이 끼어드는 순간 그 스텝의 decode가 전부 SLO를 놓쳐 「세지 않는 토큰」이 됨. 분리하면 그 토큰들이 되살아남.

버려지던 토큰이 계산에 들어옴

식당으로 치면 요리 총량은 같은데, 늦게 나가 버려지던 접시가 줄어드는 것 · 그래서 처리량이 아니라 「제때 나간 접시」가 오른다

vLLM 문서 : "Disaggregated prefill DOES NOT improve throughput." · DistServe ablation — 모든 병렬화 조합을 탐색한 colocated "vLLM++"가 기본 vLLM과 성능이 같았음.

Disaggregation은 언제 효과가 크고, 언제 작은가

공짜가 아님. KV를 인스턴스 사이로 옮겨야 하고, prefill 풀의 메모리는 놀게 됨.

효과가 큰 경우

· 입력이 길고 출력이 짧음 — prefill 비중이 커서 간섭이 심한 워크로드

· TPOT SLO가 빡빡함 — 균일한 iteration 시간이 곧 돈

· 트래픽이 이질적 — 길이가 제각각이라 청크 예산 하나로 못 덮음

· 고대역 인터커넥트 — NVLink·RDMA로 KV 전송 비용이 낮음

· 규모가 큼 — prefill:decode 비율을 실제로 조정할 만큼 노드가 많음

요약: SLO가 빡빡하고 규모가 큰 프로덕션

효과가 작거나 손해인 경우

· 처리량만 봄 — Disaggregation은 throughput을 올리지 않음. goodput 수단

· 컨텍스트가 짧음 — 간섭 자체가 작아서 얻을 게 적음

· 인터커넥트가 느림 — KV 전송 지연이 TTFT로 그대로 돌아옴

· 노드가 적음 — 풀을 나누면 각 풀의 배치가 오히려 작아짐

· prefix 적중률이 높음 — 이미 prefill이 싸서 간섭이 별로 없음

요약: 작은 배포에서는 chunked prefill이 나음

둘 중 무엇이 우월한가에 공개된 정량 비교는 아직 없다 — 워크로드로 갈린다

Session · Request · Step — 세 단위를 섞으면 다 어긋남

에이전트 워크로드를 이루는 세 겹. 어느 단위로 재느냐에 따라 같은 시스템의 수치가 달라짐.

SESSION
사람이 열고, 잘 닫지 않음
평균 62.6분

REQUEST
사람이 한 번 요청
세션당 9.2개

STEP
에이전트가 스스로 도는 한 바퀴
요청당 약 9회

「초당 몇 토큰」이 스텝 기준인지 요청 기준인지 세션 기준인지 밝히지 않으면 비교가 성립하지 않는다

출처 : Copilot 프로덕션 트레이스 arXiv:2608.00101 · TraceLab arXiv:2606.30560 (세션 평균 62.6분 · 세션당 요청 9.2개).

Agentic AI · 특징 ①

사람은 시작만 함

LLM 호출의 대부분을 사람이 아니라 에이전트 자신이 촉발함. 트래픽의 성격이 근본적으로 다름.

87%

에이전트가 스스로 촉발

Copilot 프로덕션

89.2%

같은 지표, 다른 데이터셋

TraceLab 트레이스

6.6개

사용자 메시지 1개당 자율 호출

중앙값

9.2개

세션당 요청 수

평균

단위는 토큰이 아니라 LLM 호출 건수. 사람이 한 번 말을 걸면 그 뒤로 에이전트가 예닐곱 번을 스스로 부름.

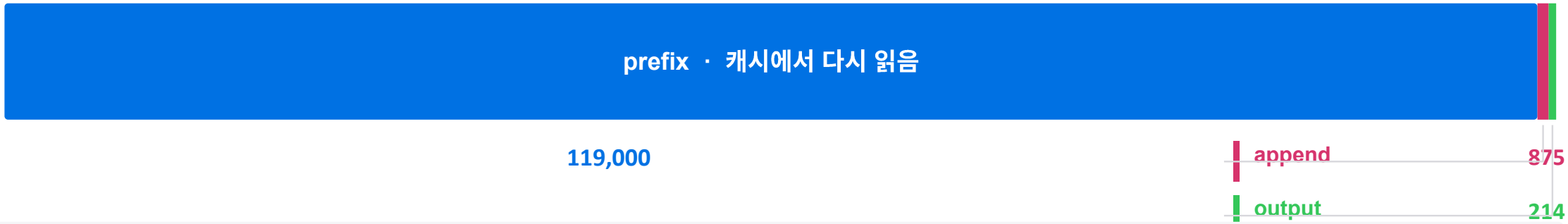
이것이 왜 중요한가 — 사람이 부르는 호출은 사람의 타이핑 속도에 묶여 도착하지만, 에이전트가 부르는 호출은 앞 호출이 끝나는 즉시 옴. 도착 패턴이 버스티해지고, 같은 세션의 컨텍스트를 계속 물고 들어옴.

출처 : Copilot arXiv:2608.00101 · TraceLab arXiv:2606.30560 — 서로 다른 두 데이터셋이 같은 지표에서 87%와 89.2%로 수렴.

Agentic AI · 특징 ③

에이전트는 만들지 않고 다시 읽음

스텝 하나의 중앙값 구성. prefix가 119,000 토큰인데 새로 프리필하는 건 875개뿐.



스텝 하나의 중앙값 구성 — 읽는 것이 새로 만드는 것의 100배가 넘음

에이전트는 만들지 않고 다시 읽음. 그래서 prefix caching이 최적화가 아니라 전제조건이 되고, 병목은 「초당 몇 토큰을 생성하나」가 아니라 「세션 상태를 얼마나 빨리 되읽나」가 됨.

이 워크로드는 모든 축이 헤비테일 — 세션 길이가 중앙값 4.2분인데 평균은 62.6분(14.9배) · 평균으로 설계하면 전부 틀린다

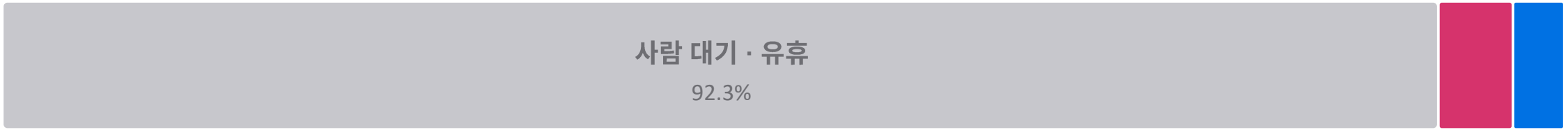
출처 : Copilot arXiv:2608.00101 · 선형 축이며 append와 output은 최소 폭으로 표기.

Agentic AI · 특징 ④

세션은 닫히지 않음

벽시계로 보면 92.3%가 사람을 기다리는 시간. 그동안 GPU 연산은 놀아도 되지만 세션의 상태는 자리를 지켜야 함.

에이전트 세션의 벽시계 시간



도구 4.8% · LLM 3.3%

25.2분

요청 경계에서 사람이 자리를 비우는 시간

중앙값

39 GB

그동안 자리를 지켜야 하는 세션 KV

Llama-70B급 GQA-FP16 · 119K

에이전틱 서빙의 1번 문제는 연산이 아니라 상태 관리다

출처 : TraceLab arXiv:2606.30560 시간 분해(유휴 무제한) · Copilot arXiv:2608.00101 요청 경계 유휴 중앙값 25.2분 · KV 39 GB는 Llama-70B급 GQA-FP16 · 119K 컨텍스트의 ▲ 환산값.

Agentic AI · 특징 ⑤

다시 읽을 뿐인데, 돈은 읽는 쪽에서 나감

비용을 쓰는 곳은 「생성」이 아님. 세션 비용의 9할이 입력 쪽.

59.5%

prefix · 다시 읽기

정가 기준

29.2%

append · 새로 프리필

11.2%

output · 생성

88.7%

입력이 가져가는 비용

prefix + append

토큰 경제학의 직관이 뒤집히는 지점. 「출력 토큰이 비싸다」는 가격표는 맞지만, 에이전트 워크로드에서 실제로 청구서를 채우는 것은 압도적으로 입력 쪽.

그래서 최적화의 순서도 바뀜 — 생성을 빠르게 하는 것보다 이미 있는 것을 다시 읽지 않게 하는 것이 먼저. 캐시가 살아 있어도 스텝 시간의 3분의 1이 프리필이고, 캐시가 죽으면 그 비율이 통째로 올라감.

출처 : Copilot arXiv:2608.00101 · 세션 비용 구성은 정가 기준.

Recompute vs. Restore — 유휴 25분 동안 39 GB를 어떻게 할 것인가

버릴 것인가, 붙잡을 것인가, 아니면 되살리는 시간을 연산 뒤에 숨길 것인가.

버리고 재계산

11.1초

$0.074

기준

재계산 대비 시간

저장 → 복원

1.5초

$0.013

÷ 5.6

재계산 대비 시간

복원 + 연산 겹침

가려짐

$0.013

시간이 사라짐

재계산 대비 시간

유휴 25분 동안 39 GB를 어떻게 할 것인가 — 이것이 에이전틱 서빙의 1번 문제

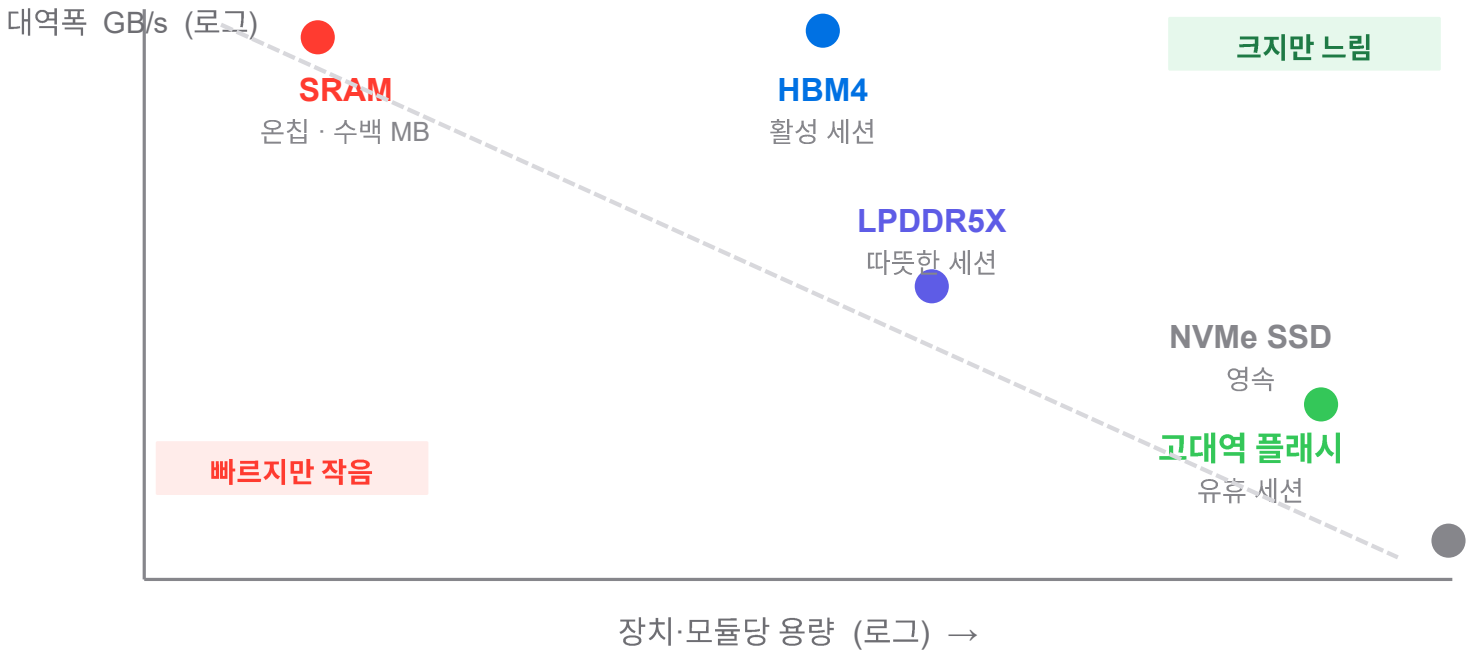
연산이 아니라 상태 관리. 그리고 답은 「버리기」와 「붙잡기」 사이 어딘가에 있음.

세 번째 열이 핵심 — 복원을 다음 스텝의 연산과 겹치면 시간이 가려진다 · 필요한 것은 더 빠른 스토리지가 아니라 「언제 되살릴지 아는 스케줄러」다

▲ 파생 비교 · 재계산은 119K 프리필을 다시 도는 비용, 복원은 39 GB를 스토리지에서 되읽는 비용 기준.

한 가지 메모리로는 안 됨

활성 세션은 대역폭 축에, 유휴 세션은 용량 축에 놓임. 두 축을 한 소재로 동시에 만족시킬 수 없음.



한 종류로는 대응이 불가능함 — 세션의 상태에 따라 층을 옮겨 다녀야 함

활성 세션

지금 토큰을 뱉고 있는 세션. 스텝마다 KV 전체를 읽으므로 대역폭이 전부.

따뜻한 세션

몇 분 안에 돌아올 세션. 용량이 크고 적당히 빠른 층에 둠.

유휴 세션

25분씩 자리를 비우는 세션. 용량 단가가 지배하고 복원 지연은 겹쳐서 숨김.

그래서

메모리 계층화는 비용 절감이 아니라 에이전틱 워크로드의 구조적 요구사항.

활성은 대역폭 축, 유휴는 용량 축 · 두 축을 한 소재로 동시에 만족시킬 수 없다는 것이 메모리 계층화의 근거

점의 위치는 공개된 장치 스펙의 자릿수 감각 — 정밀한 벤치마크가 아님.

병목은 옮겨 다님.

compute → memory → power

그리고 그때마다 답이 달라짐

AI 반도체 · 지형

설계는 두 갈래로 갈림

메모리를 어디에 두느냐. 이 선택 하나가 ridge point를 세 자릿수 차이로 벌림.

SRAM에 다 올림

Cerebras · Groq

메모리를 칩 위에 둠. 대역폭이 PB/s 단위로 터짐.

그래서 ridge point가 한 자릿수 — 배치 1에서도 연산기가 바쁨. 저지연 서빙에 구조적으로 유리함.

대가는 용량. 44 GB(Cerebras) · 500 MB(Groq)로는 큰 모델을 한 칩에 못 올림. 수십~수백 칩에 쪼개야 하고, 그러면 칩 간 통신이 새 병목이 됨.

ridge 0.65 ~ 8

HBM을 붙임

NVIDIA · AMD · Google · SambaNova

용량을 먼저 확보함. 192~288 GB를 한 패키지에 붙임.

큰 모델과 긴 KV를 담을 수 있음. 생태계·소프트웨어가 성숙해 있음.

대가는 ridge point. 세대마다 연산은 배로 뛰는데 대역폭은 그만큼 못 따라감. 바이트 하나를 읽어 갚아야 할 연산량이 계속 비싸짐 — 이게 메모리 벽.

ridge 280 ~ 1,800

ridge point = dense peak FLOPS ÷ 메모리 대역폭 · 낮을수록 「바이트가 싸고 연산이 귀한」 칩, 높을수록 「연산이 싸고 바이트가 귀한」 칩

AI 반도체 · 회사별

전용 실리콘 진영 ①

대역폭으로 승부함. LLM decode가 memory-bound라는 사실을 하드웨어로 정면 공략한 설계들.

웨이퍼 스케일

Cerebras
WSE-3 Turbo · CS-4

웨이퍼 한 장이 통째로 칩. 900,000코어 · 4조 트랜지스터.

◎ 44 GB SRAM에 43.2 PB/s — HBM보다 세 자릿수 빠름
◎ ridge 0.65 — 배치 1에서도 연산기가 바쁨

◎ MemoryX로 외부 1.5 TB~1.2 PB를 붙여 가중치를 스트리밍

× 웨이퍼 위 상주 용량은 44 GB — 긴 KV를 올릴 자리가 없음
× 오프-웨이퍼 I/O가 2.4 Tb/s로 다섯 자릿수 절벽
× 헤드라인 PFLOPS에 sparse 주석 · dense는 미공개

결정적 데이터플로

Groq 3 LPU
LP30 · LPX 랙

컴파일러가 모든 데이터 이동을 정적으로 스케줄링함. 캐시도 스케줄러도 없음.

◎ 500 MB SRAM에 150 TB/s — HBM4의 약 7배
◎ 결정적(deterministic) 실행 → 지연 편차가 거의 없음
◎ ridge 8 — 저지연 구간의 챔피언

× 칩당 500 MB. 모델 하나에 수백 칩이 필요함
× FP16/BF16 성능 미공개

RISC-V 오픈

Tenstorrent
Blackhole · Galaxy

RISC-V 메시. 120 Tensix 코어 + 16 SiFive X280. 스택이 전부 오픈소스.

◎ 개방형 — tt-metal / tt-forge를 직접 고칠 수 있음
◎ GDDR6라 칩·시스템 단가가 낮음
◎ 카드 단위로 구매 가능 (p150a 32 GB)

× 칩당 512 GB/s — HBM 대비 대역폭이 한 자릿수 낮음
× ridge 1,297 — decode에서 불리한 좌표
× 차세대 Quasar는 아직 출하 전

◎ = LLM 추론 관점의 강점, × = 약점. 2026-08 공개 자료 기준. FLOPS는 전부 dense.

AI 반도체 · 회사별

전용 실리콘 진영 ②

용량과 생태계로 승부하거나, 아예 트랜스포머만 굽거나.

클라우드 전용

Google TPU
TPU 8i · 8세대

2026-04 Cloud Next에서 8세대 공개. 학습용 8t와 추론용 8i로 나뉨.

◎ 288 GB HBM · 8.6 TB/s — TPU 중 대역폭 최고
◎ 온칩 SRAM 384 MB(3배) — KV를 다이에 올리려는 설계
◎ ICI 19.2 Tb/s(2배) · Boardfly로 1,152칩

× 아직 GA 아님 — 「올해 안」이라고만 발표
× Google Cloud 전용 · JAX/XLA 중심
× ridge 1,174는 FP4 기준

데이터플로

SambaNova
SN50 RDU

재구성 가능 데이터플로. 3단 메모리(SRAM–HBM–DDR)로 큰 모델을 소켓 하나에 얹음.

◎ 432 MB 온칩 SRAM + 64 GB HBM + 대용량 DDR5
◎ 한 소켓에 여러 모델을 상주시키고 즉시 전환
◎ dense FP8 3.2 PFLOPS — 소켓당 연산은 넉넉함

× HBM 대역폭 1.8 TB/s — 세대 대비 낮음
× ridge 1,778 — 이 목록에서 가장 연산 편중
× FP4를 저장에만 쓰고 연산은 FP8 경로 → FLOPS 이득 없음

트랜스포머 전용

Etched
Sohu → 랙 제품

트랜스포머만 하드와이어한 ASIC. 다른 아키텍처는 아예 못 돌림.

◎ 구조를 고정한 만큼 이론적 효율 상한이 높음
◎ prefill/decode를 칩 수준에서 분리한 랙 구성
◎ 2026-08 첫 고객 인도(Jane Street) 발표

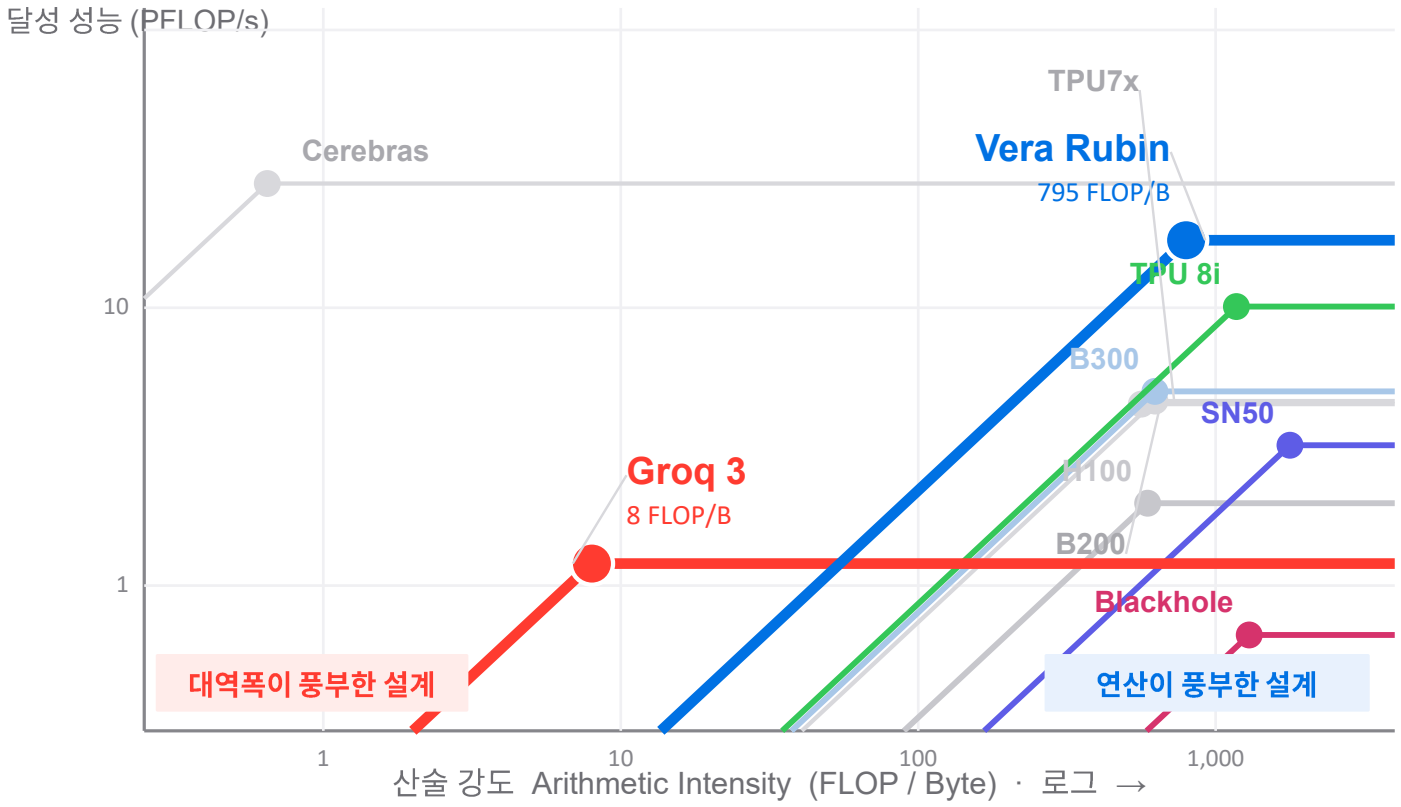
× 메모리 용량 · 대역폭 · FLOPS 모두 [데이터 없음]
× 공개된 수치가 벤더 주장 하나뿐이라 검증 불가
× 구조가 바뀌면 칩이 무용지물 — MLA·SSM 등

⚠ Etched만 스펙 비공개 — 2024년 Sohu 발표의 144 GB HBM3e 외에는 1차 출처가 없어 roofline에 올리지 못했다 · 「없어서 못 그린 것」이지 「나빠서 뺀 것」이 아니다

AI 반도체 · Roofline

한 장의 Roofline 위에 전부 올려놓으면

ridge point가 0.65에서 1,778까지 2,700배 벌어짐. 같은 문제를 정반대로 푼 것.



Vera Rubin ridge 795

288 GB HBM4 · 22 TB/s · dense FP8 17.5 PF. 용량과 연산을 동시에 밀었고, 그 대가로 ridge가 오른쪽으로 갔음.

Groq 3 LPU ridge 8

500 MB SRAM · 150 TB/s · dense FP8 1.2 PF. 대역폭을 두 자릿수 올려 ridge를 왼쪽 끝으로 끌어왔음.

decode는 왼쪽

AI가 배치 크기. 배치 32면 AI 32 — Groq의 지붕에는 닿고 Vera Rubin의 지붕에는 한참 못 미침.

그래서 결론이 갈림

저지연 소배치는 SRAM 진영이, 대용량·고배치는 HBM 진영이 유리함. 「어느 칩이 빠른가」는 좌표 없이는 답이 없음.

per-chip · dense · peak 스펙 · Cerebras는 FP16 dense 추정값(EST 28 PF/웨이퍼), 나머지는 8비트급 · Etched는 공개 수치가 없어 제외.