LLM inference speedup

YSDA

by Roman Gorb

Кто я такой?


ДОКТОР
ЛИВСИ
ОЧЕНЬ ХОРОШИЙ И ВЕСЕЛЫЙ
ЧЕЛОВЕК.
ХАРАКТЕР ОБЩИТЕЛЬНЫЙ.
НЕ ЖЕНАТ.

Кто я такой?

Роман Горб

- YandexGPT (3+ года):
  - PEFT
  - Продуктовый ML
  - Ускорение инференса
- R&D-лаба Huawei (1 год)
- Закончил ФизТех, ШАД
- Призер ВСОШ по математике и информатике
- 1 взр. разряд по волейболу =)

Agenda

- Motivation
- Tradeoff
- LLM inference
- Speculative Decoding
- Quantization
- Knowledge Distillation

Motivation

Scenarios

Models:

- Decoder-only, e.g GPT
- Enc-Dec, e.g. UL2
- Small vs large
  - Compute vs memory bound
  - OOM or not
- Batch size
- SFT vs PEFT

Tasks:

- Generative:
  - Q&A
  - Dialogs
  - Summary
- Discriminative:
  - Classification
  - Regression
  - NER

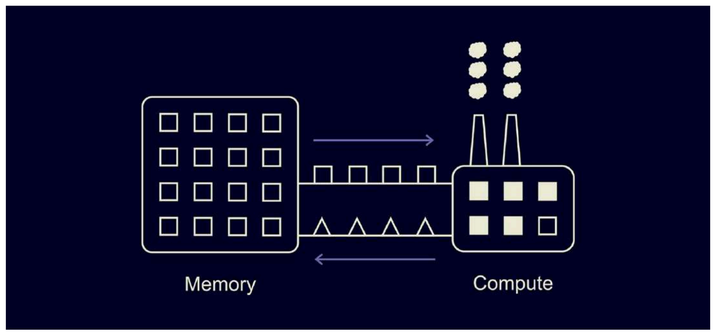
Stages:

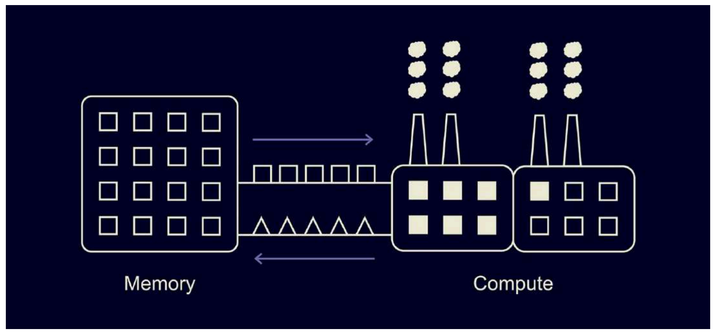
- Context processing
- Decoding

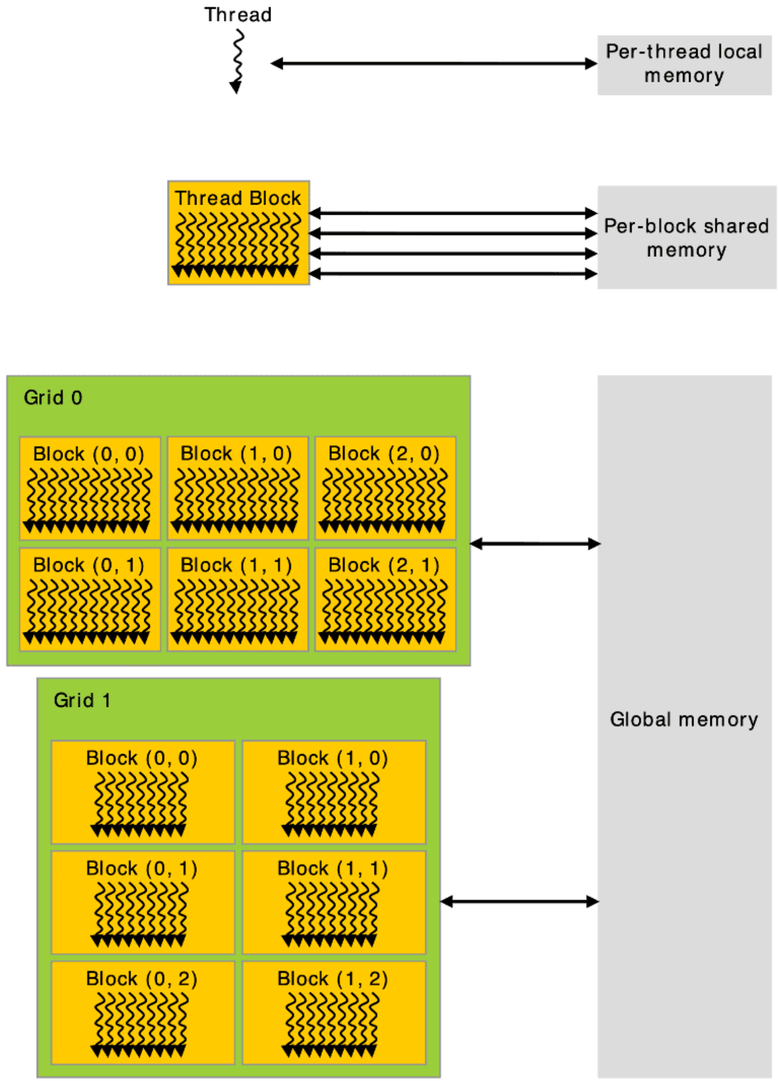
GPU

Architecture and limirations

- Core types: CUDA vs Tensor
- Memory transfer and compute
- VRAM is 10x slower than SM
- VRAM=80GB, SM=228KB





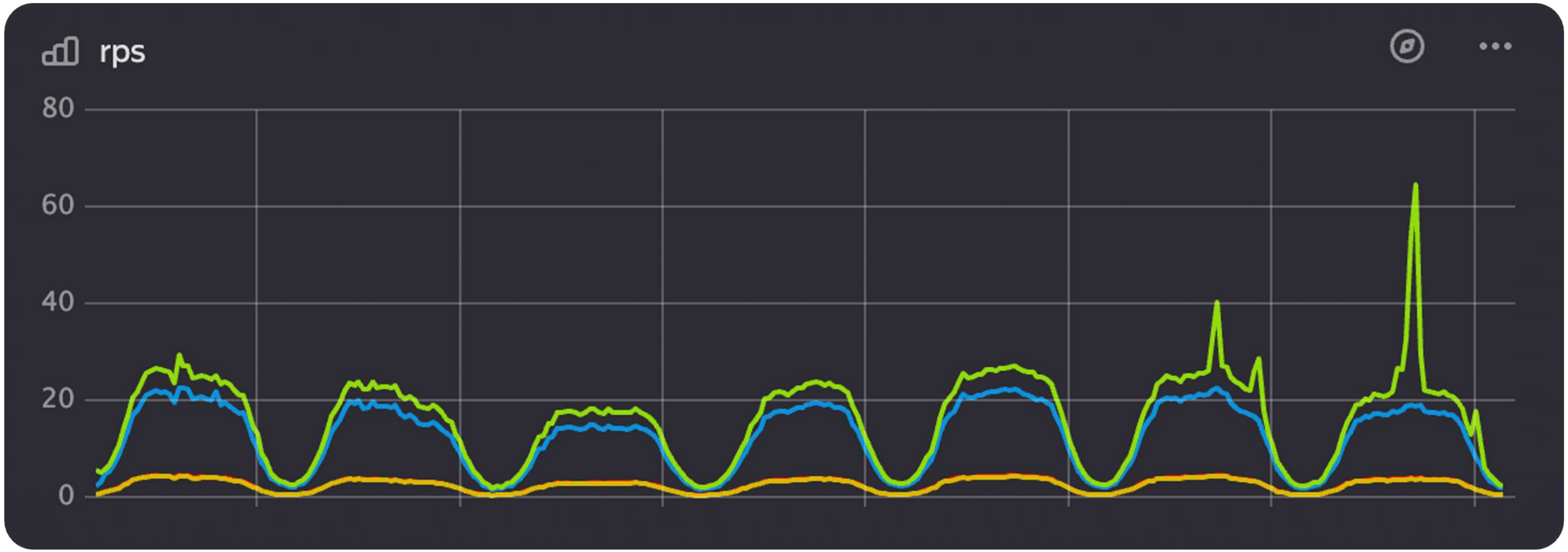


* https://habr.com/en/companies/yandex/articles/672396/

Production

Compute budget

- RPS — The number of HTTP-style requests completed per second
- Latency — The amount of time (in microseconds) taken for a single request to complete



Hands-on case

2K rps
service load

<3
sec per request — latency limit

↓

20%
cache hit

↓

1,6K (1,8K) rps
on avg (at peak)

1 ⟷ 6 rps
GPU

300
GPU is service demand

200
GPU is computational budget

↓

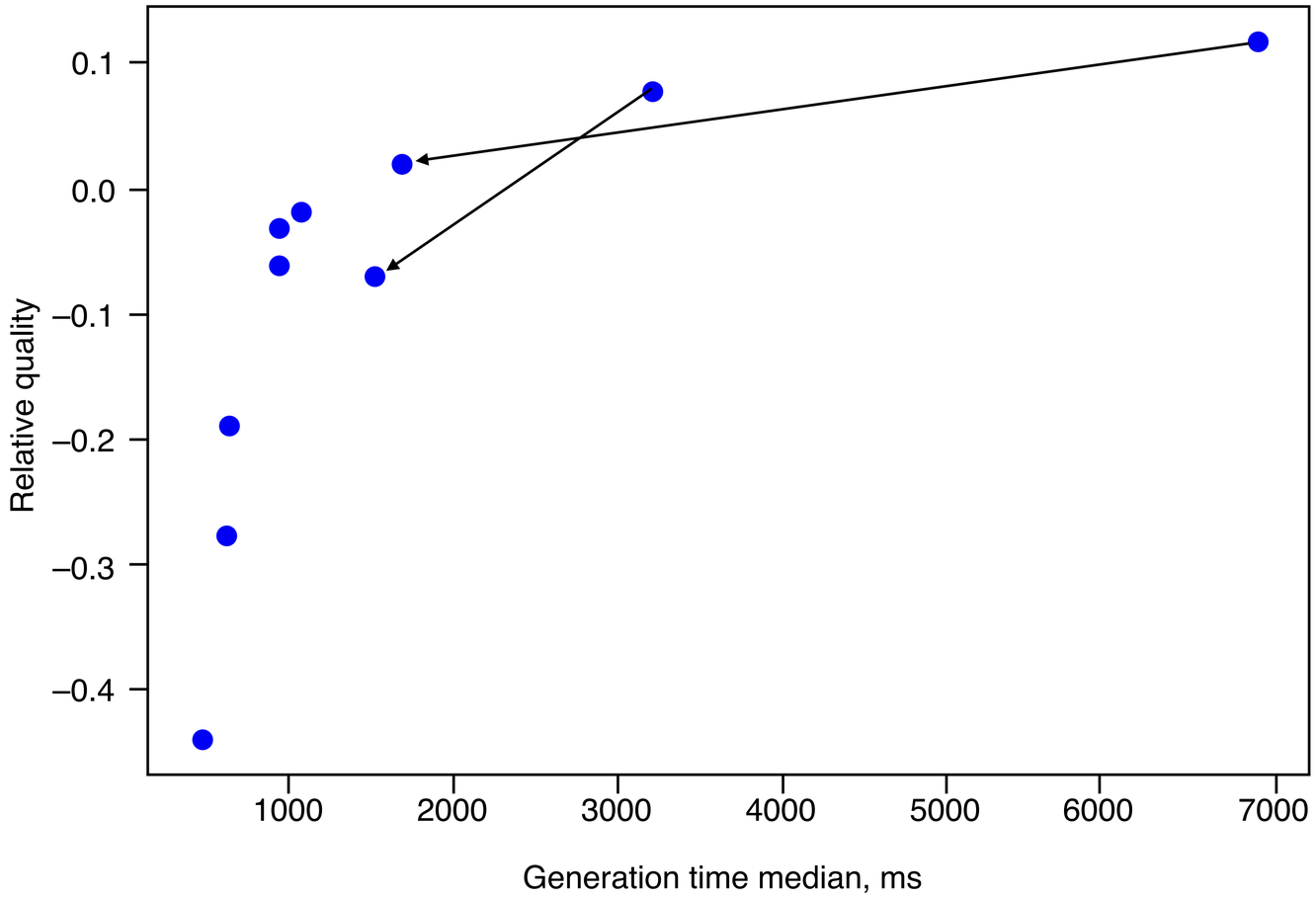
Oops

Tradeoff

Pareto curve

Quality vs Compute

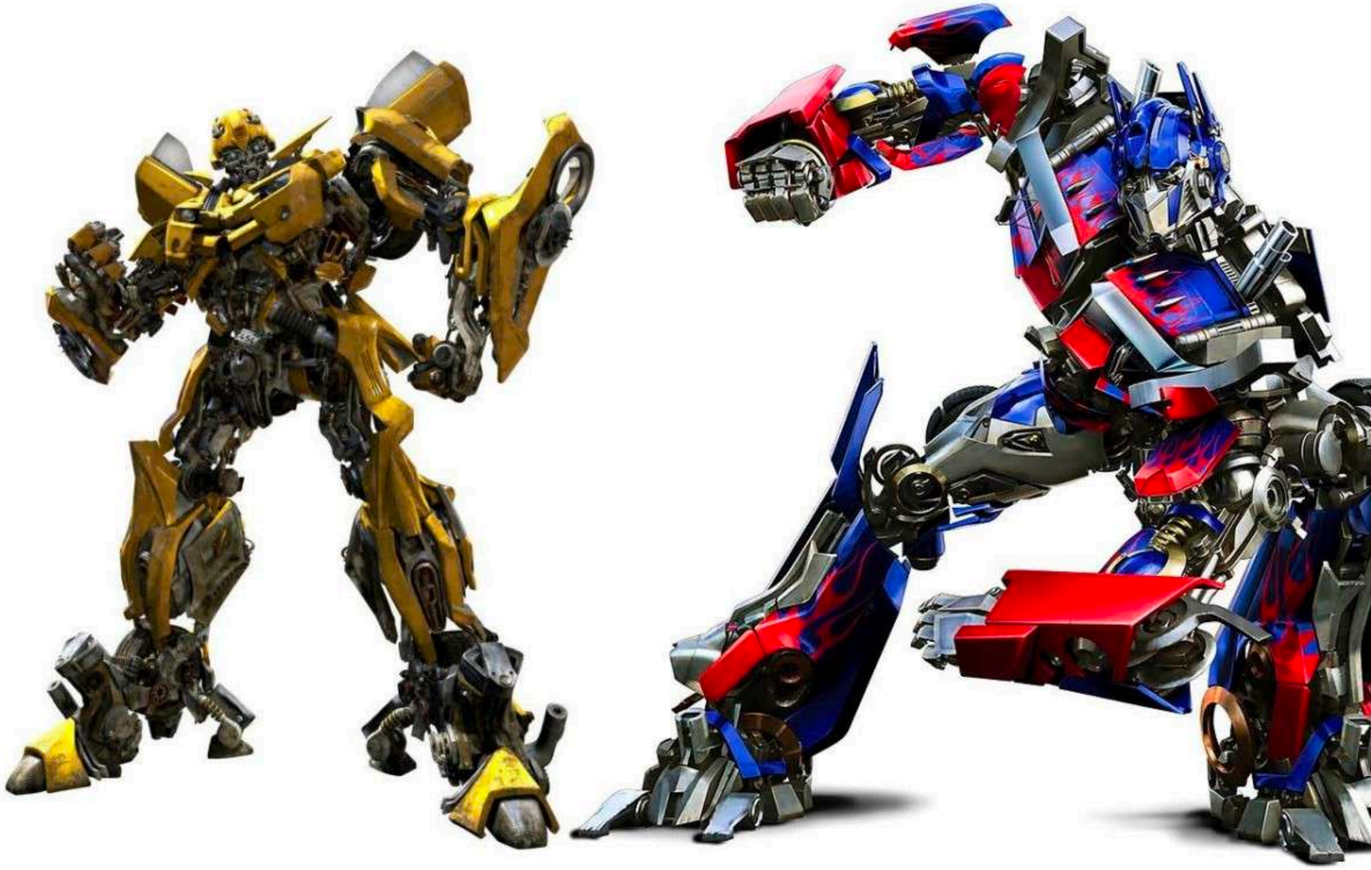
- Latency or RPS for x axis
- Quality metric for y axis
- Inflection point is optimal
- Should consider compute budget conditions



Challenge

How to add new points?

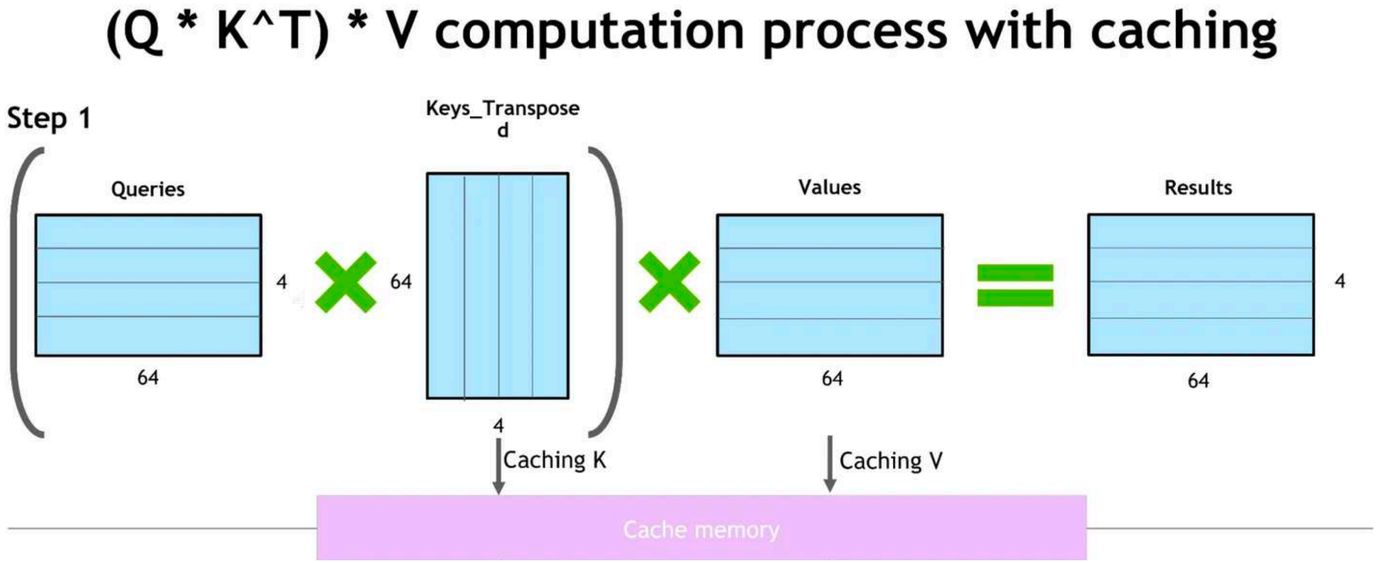
LLM inference



Classification

or context processing

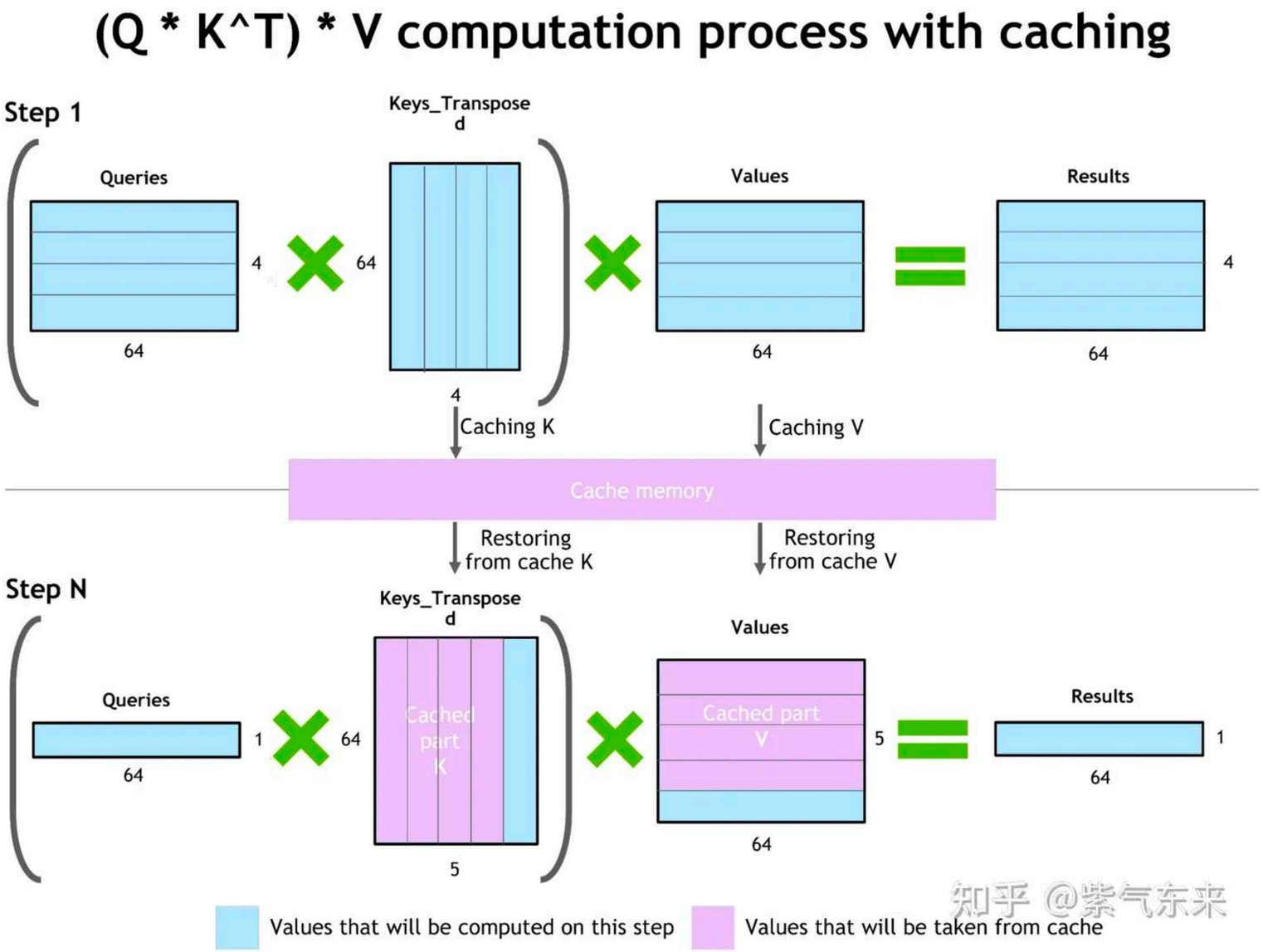
- *Attention* with squared mask
- 128k tokens
- 1 forward
- GeMM
- Compute bound



Generation

autoregressive

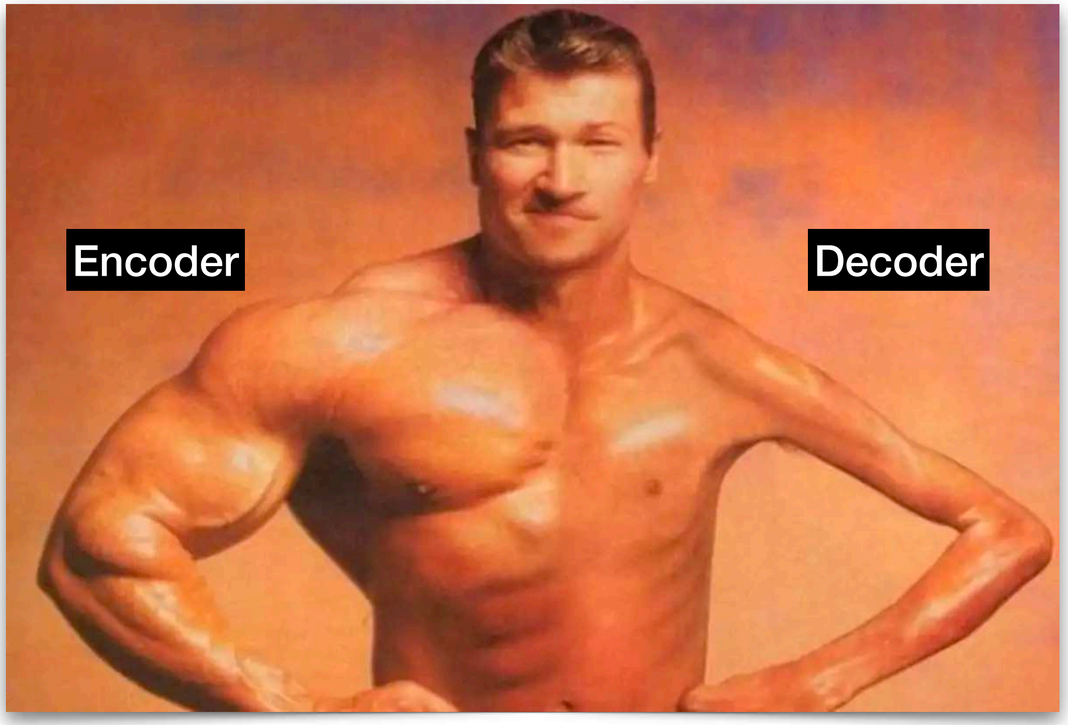
- Triangle mask
- 1 forward per token
- GeM**V**
- Loading weights on each step
- Memory bound



Encoder-decoder

breaking the rules

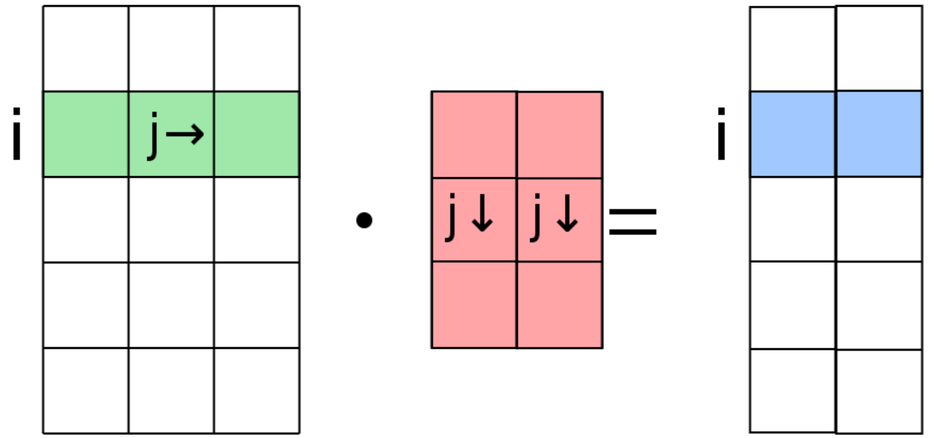
- 1 forward encoder-a for context
- Cross-Attention
- N forward-ов decoder-a per token
- Easy strategy to reduce cost:
  - 70% to encoder
  - 30% to decoder



Road to compute bound

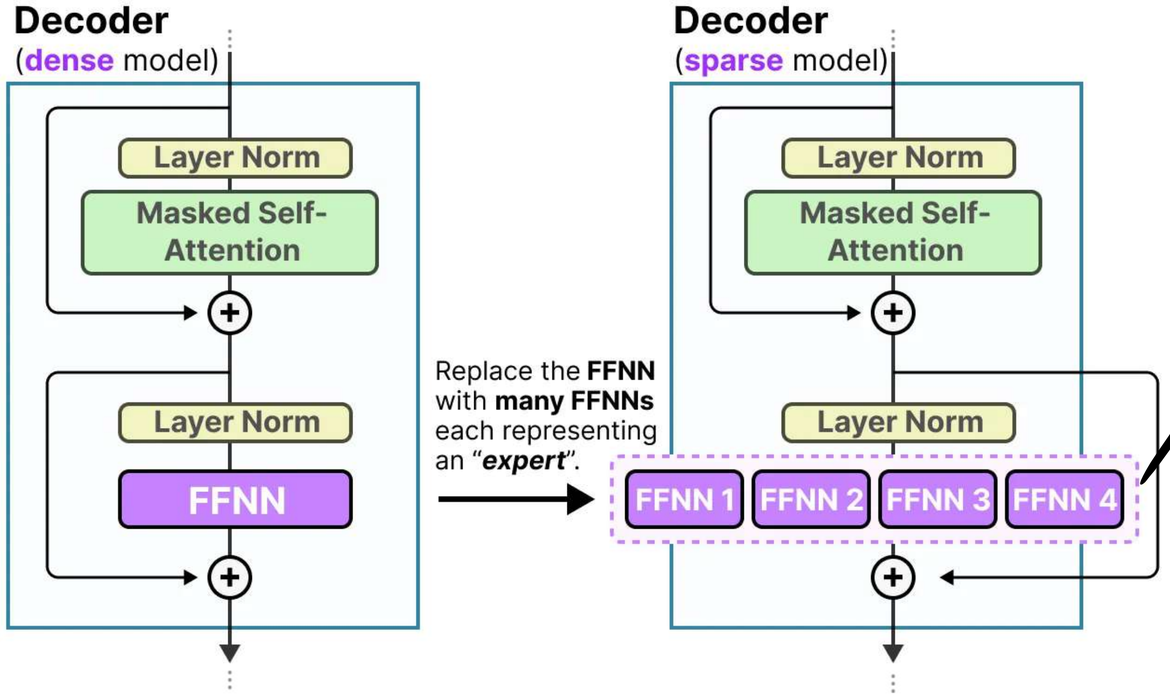
Size matters

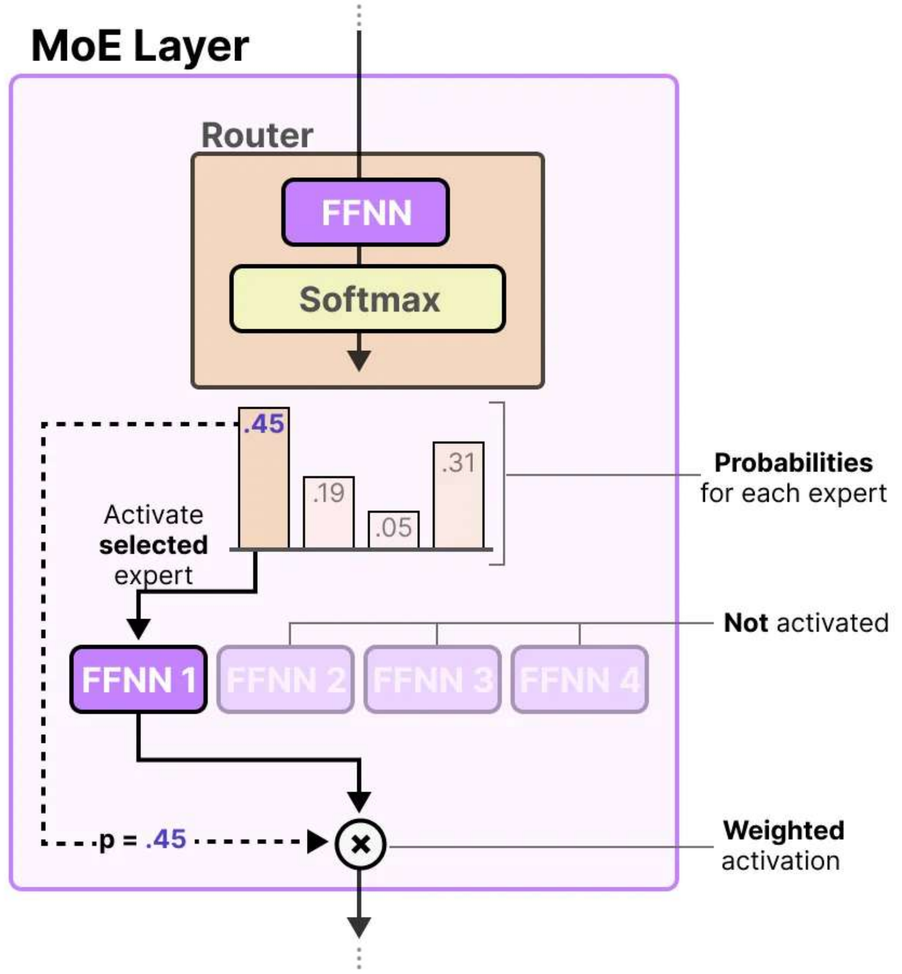
- batch_size=N:
  - Encoder/context: no change
  - Decoder: matrix x N vectors
  - => For large N similar to GeMM
- Small model size:
  - => not bounded memory transfer
  - => compute bound

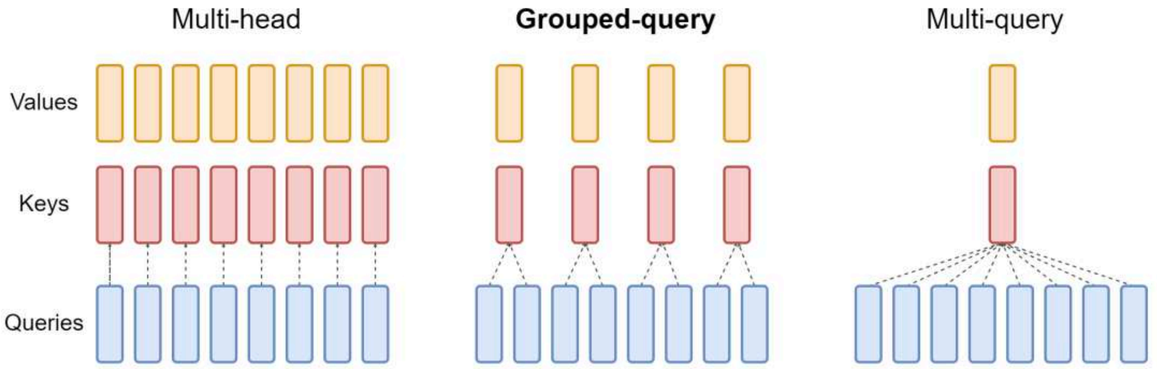


Road to compute bound

Architecture tweaks



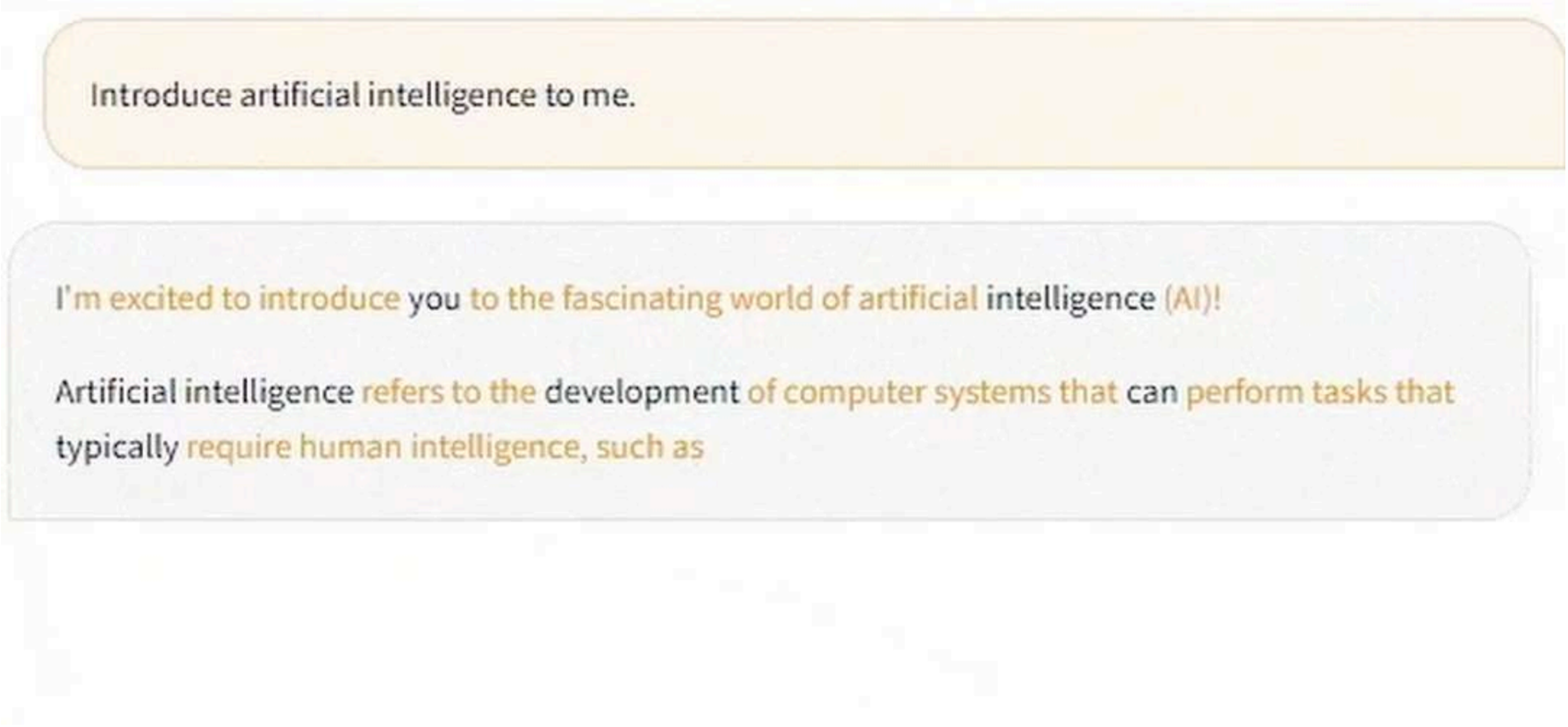




Speculative Decoding

Speculative Decoding

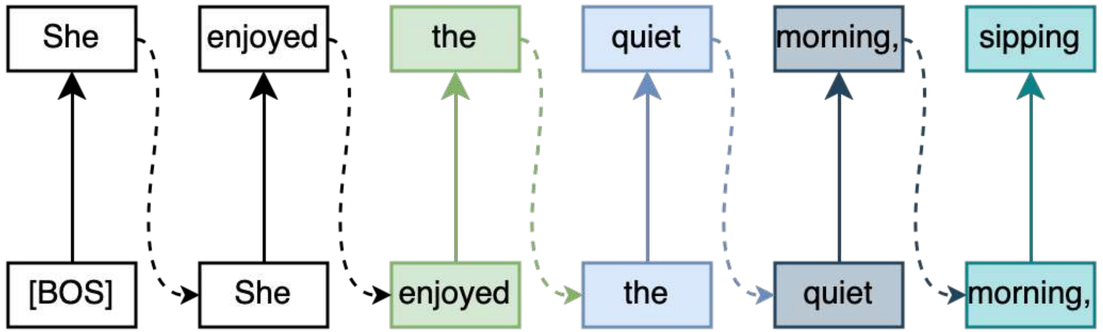
idea



Speculative Decoding

idea

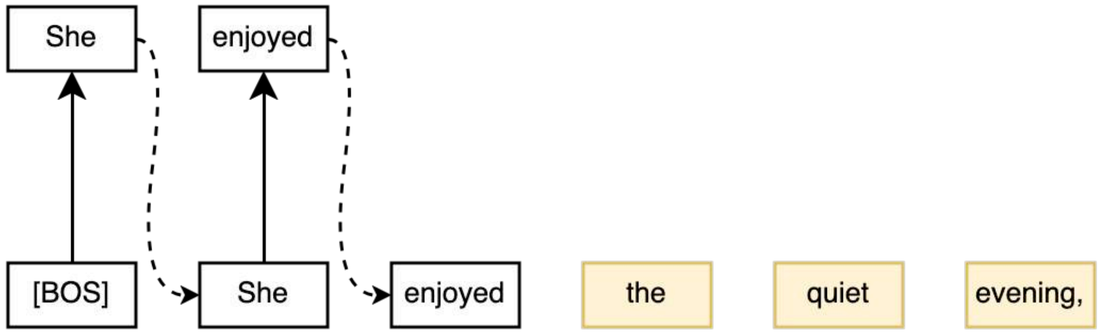
- Decoding
  - 1 forward per token



Speculative Decoding

idea

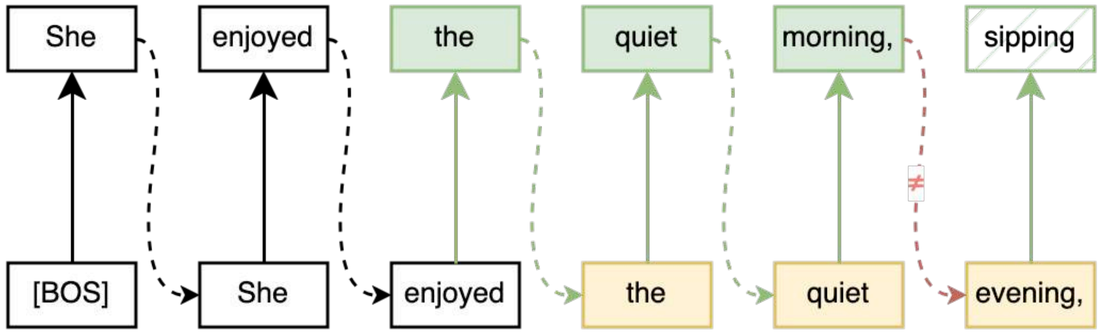
- Decoding
  - 1 forward per token
- Speculative Decoding
  1. Small model generates K tokens



Speculative Decoding

idea

- Decoding
  - 1 forward per token
- Speculative Decoding
  1. Small model generates K tokens
  2. Large model verifies



Speculative Decoding

Formal part

Algorithm 2 Speculative Sampling (SpS) with Auto-Regressive Target and Draft Models

Given lookahead K and minimum target sequence length T.
Given auto-regressive target model $q(.|.)$, and auto-regressive draft model $p(.|.)$, initial prompt sequence $x_0, \ldots, x_t$.
Initialise $n \leftarrow t$.
while $n < T$ **do**
  for $t = 1 : K$ **do**
    Sample draft auto-regressively $\tilde{x}_t \sim p(x|, x_1, \ldots, x_n, \tilde{x}_1, \ldots, \tilde{x}_{t-1})$
  end for
  In parallel, compute $K + 1$ sets of logits from drafts $\tilde{x}_1, \ldots, \tilde{x}_K$:

$$q(x|, x_1, \ldots, x_n),\ q(x|, x_1, \ldots, x_n, \tilde{x}_1),\ \ldots,\ q(x|, x_1, \ldots, x_n, \tilde{x}_1, \ldots, \tilde{x}_K)$$

  for $t = 1 : K$ **do**
    Sample $r \sim U[0, 1]$ from a uniform distribution.
    if $r < \min\left(1, \frac{q(x|x_1,\ldots,x_{n+t-1})}{p(x|x_1,\ldots,x_{n+t-1})}\right)$, **then**
      Set $x_{n+t} \leftarrow \tilde{x}_t$ and $n \leftarrow n + 1$.
    else
      sample $x_{n+t} \sim (q(x|x_1, \ldots, x_{n+t-1}) - p(x|x_1, \ldots, x_{n+t-1}))_+$ and exit for loop.
    end if
  end for
  If all tokens $x_{n+1}, \ldots, x_{n+K}$ are accepted, sample extra token $x_{n+K+1} \sim q(x|, x_1, \ldots, x_n, x_{n+K})$ and set $n \leftarrow n + 1$.
end while

Speculative Decoding

speedup

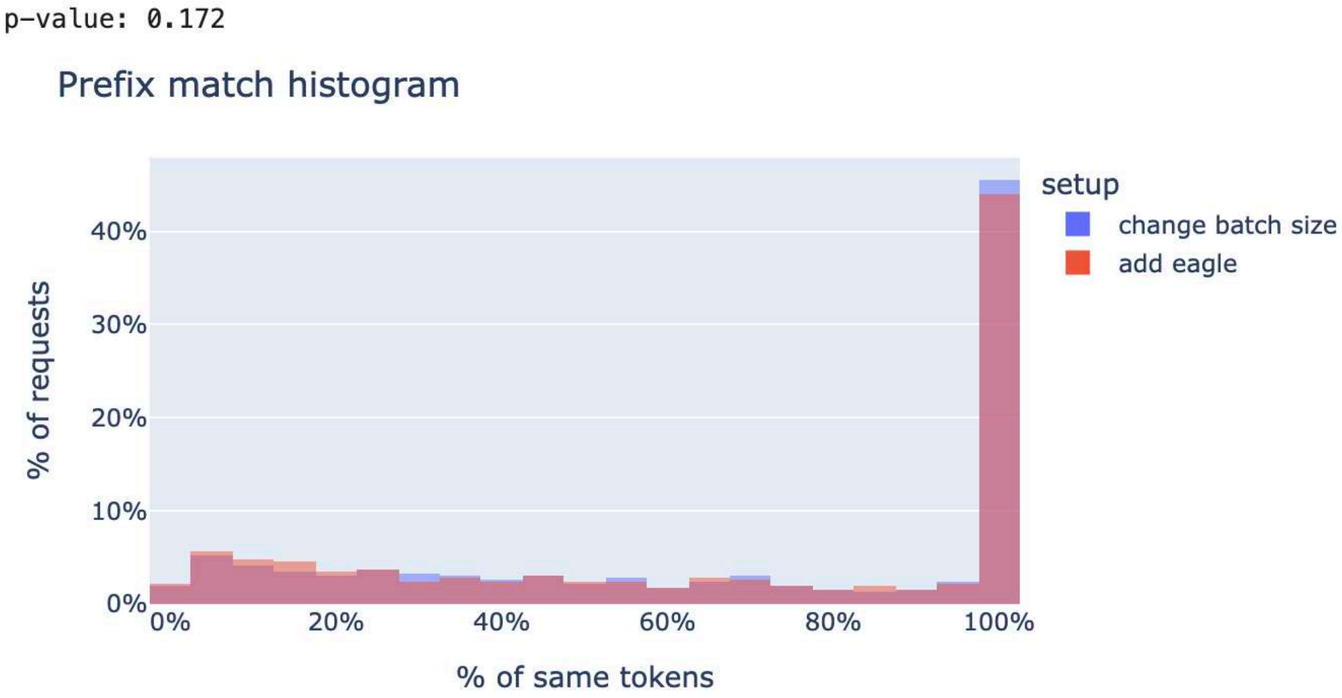
- Decoding:
  - 1 forward per each token
- Speculative Decoding:
  - 1 verifier’s forward per k tokens + k drafter’s forwards
- Implications:
  - less forwards of large model
  - same cost for forwards on 1 and k tokens

# Токенов	Время
1	14.0 ms
6	14.1 ms

Speculative Decoding

Quality

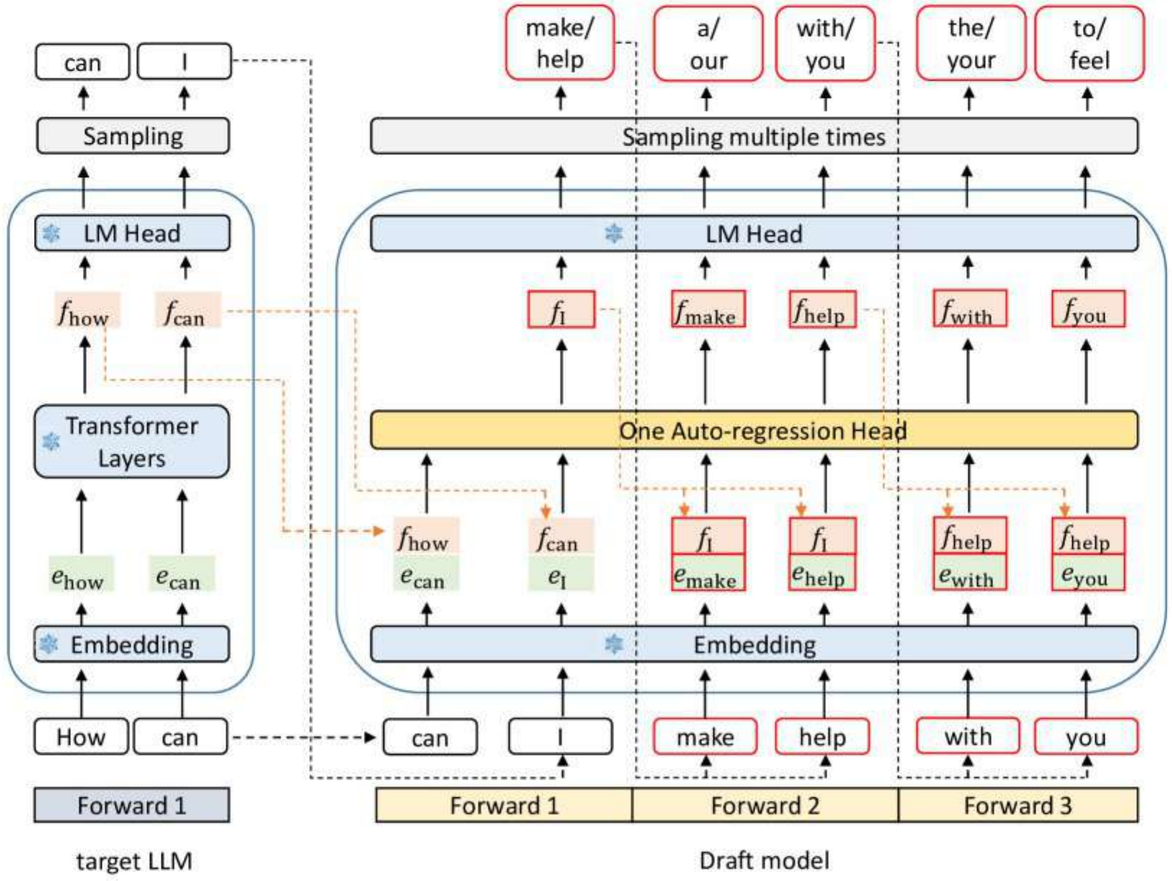
- Полноценная приёмка это дорого
- Преимущество SpecDec:
  - Математически – генерации те же
  - Погрешности как при смене batch_size

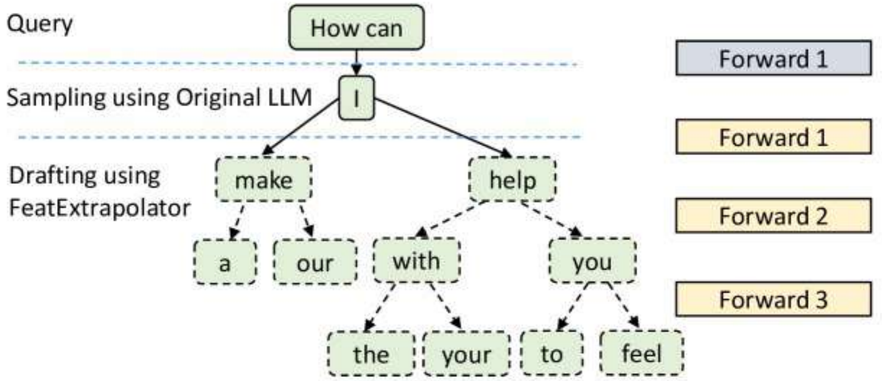


EAGLE

method

- Drafter is a head instead of model
- Frozen base model's body🧊
- High AccRate
- Head – 1-layer transformer over hiddens of base model
- Tree of hypotheses





EAGLE

why

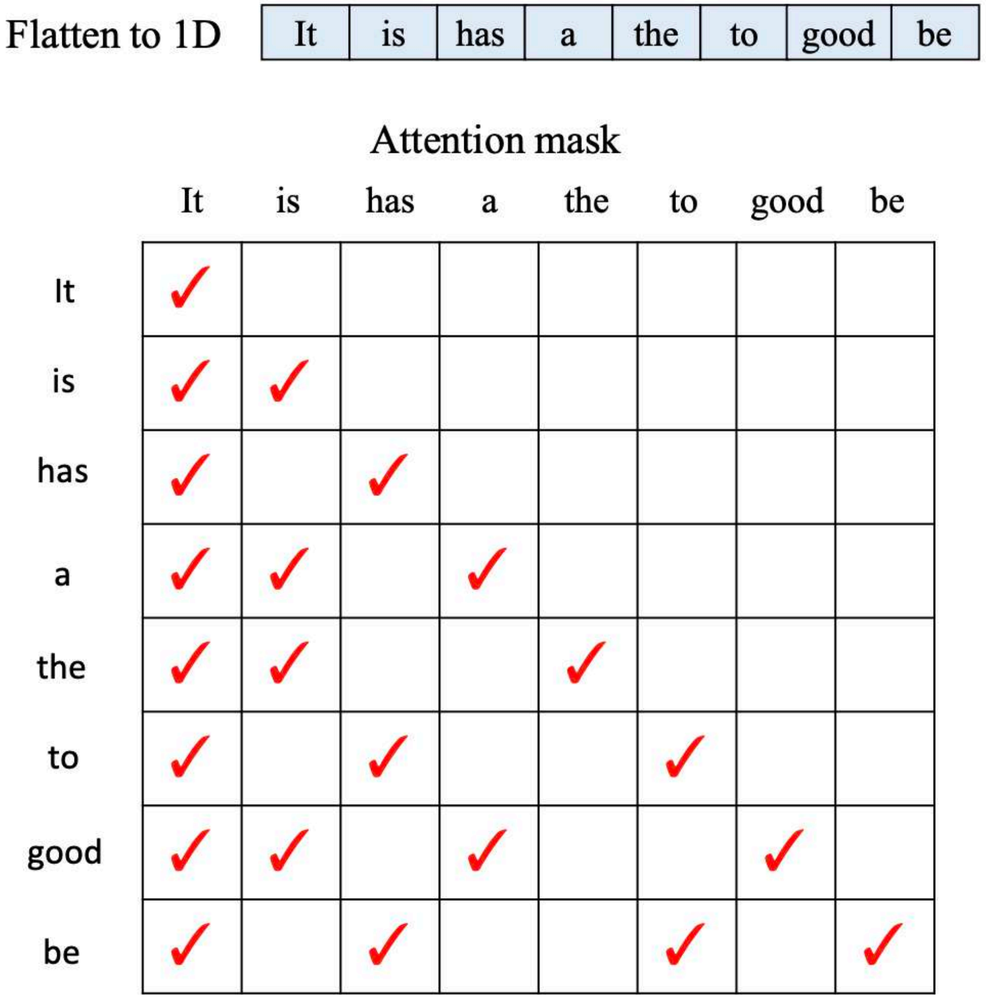
- First method for large batch_size
- Low overhead:
  - lightweight head (1/48)
- High acc. rate:
  - hidden of model
  - sequential mode

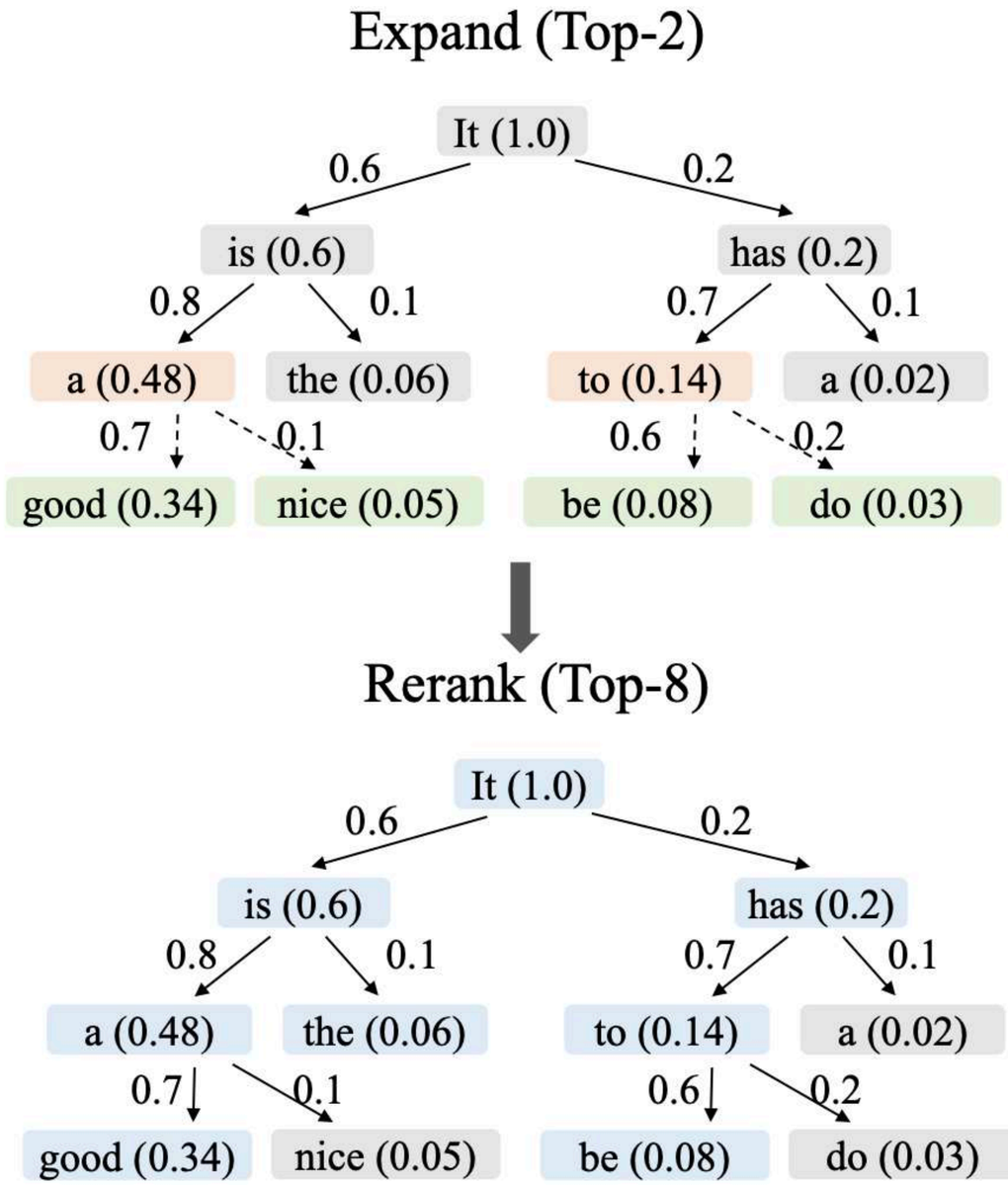
Models	#Mean Accepted Tokens	Overall
EAGLE2 🥇	**4.36**	**2.25x**
EAGLE 🥈	3.57	2.16x
SpS 🥉	2.29	1.83x
Hydra	3.26	1.80x
PLD	1.74	1.55x
Medusa	2.32	1.44x
REST	1.63	1.32x
Lookahead	1.65	1.08x

https://github.com/hemingkx/Spec-Bench/blob/main/Leaderboard.md

EAGLE-2

upgrade





Quantization

Quantization

idea

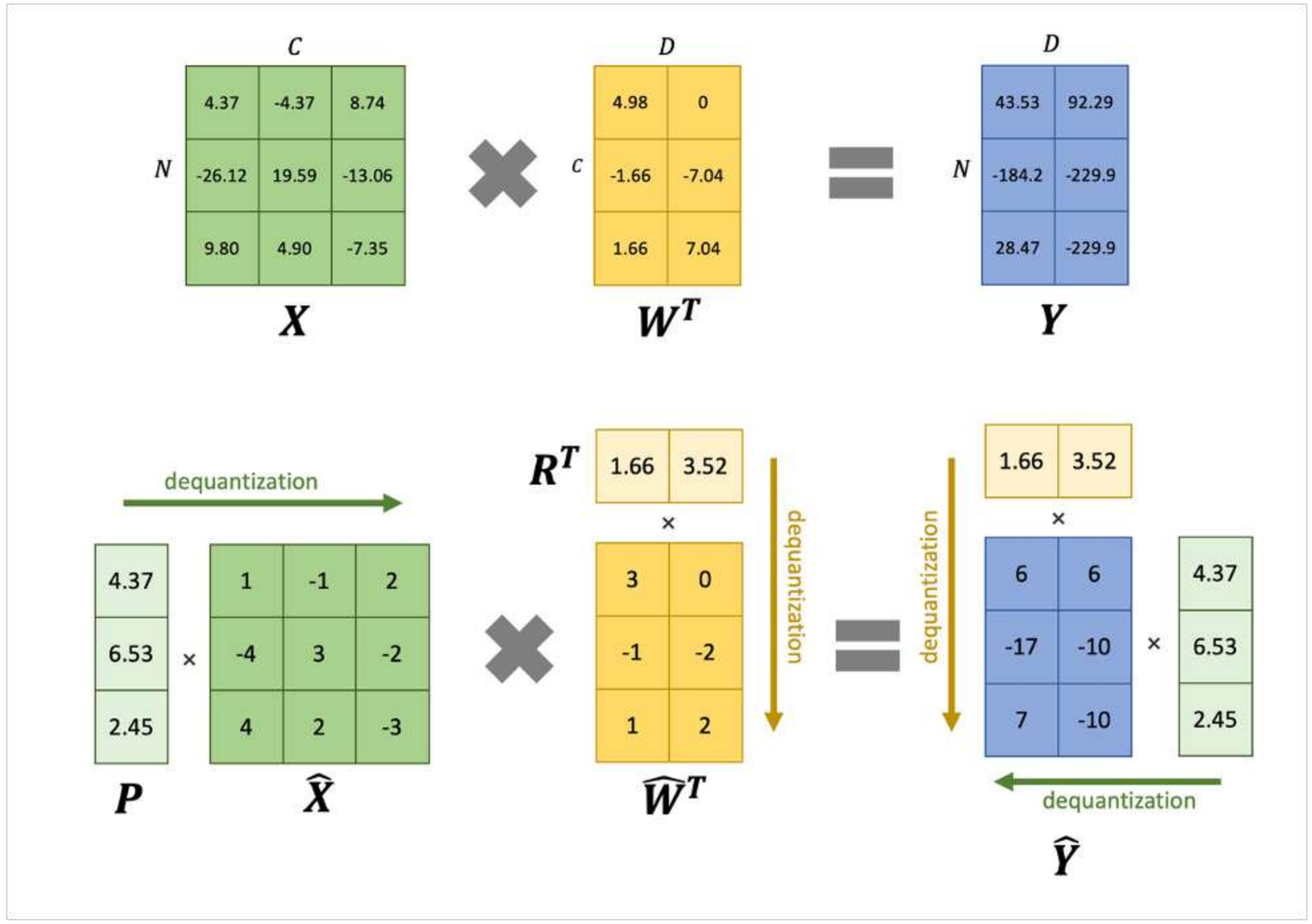
Quantizing a real-valued tensor $\mathbf{x}$ is performed by first mapping it to an unsigned integer grid:

$$\mathbf{x}^{(\mathbb{Z})} = \text{clip}\left(\left\lfloor \frac{\mathbf{x}}{s} \right\rceil + z; 0, 2^b - 1\right), \qquad (1)$$

It is possible to approximately recover the real-valued input $\mathbf{x}$ through an operation that is often referred to as *de-quantization*:

$$\widehat{\mathbf{x}} := q\left(\mathbf{x};\, s, z, b\right) = s\left(\mathbf{x}^{(\mathbb{Z})} - z\right) \approx \mathbf{x}. \qquad (2)$$

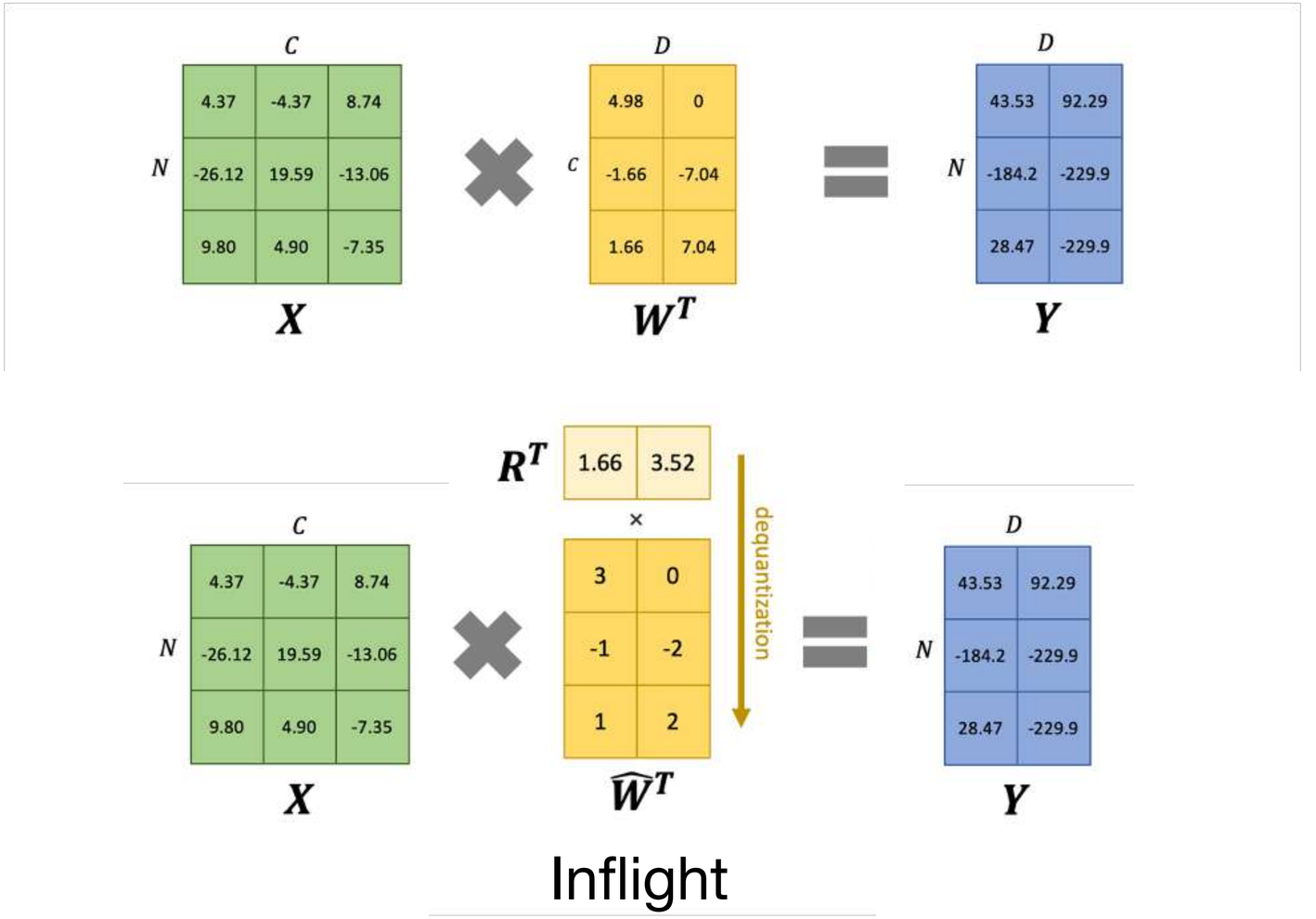
In the case of symmetric quantization, we restrict the quantization grid to be symmetric around z.



Quantization

types

- weight-only (wNa16)
  - reduce memory footprint
  - speedup memory bound
- wNaM
  - **also** reduce compute
  - speedup compute bound



Stages
and objectives

- Quant: preserve benchmarks quality
- Backend support: gain speedup
- Could be separated
- But highly interconnected

OPT-175B	LAMBADA	HellaSwag	PIQA	WinoGrande	OpenBookQA	RTE	COPA	**Average↑**	**WikiText↓**
FP16	74.7%	59.3%	79.7%	72.6%	34.0%	59.9%	88.0%	66.9%	10.99
W8A8	0.0%	25.6%	53.4%	50.3%	14.0%	49.5%	56.0%	35.5%	93080
ZeroQuant	0.0%*	26.0%	51.7%	49.3%	17.8%	50.9%	55.0%	35.8%	84648
`LLM.int8()`	74.7%	59.2%	79.7%	72.1%	34.2%	60.3%	87.0%	66.7%	11.10
Outlier Suppression	0.00%	25.8%	52.5%	48.6%	16.6%	53.4%	55.0%	36.0%	96151
SmoothQuant-O1	74.7%	59.2%	79.7%	71.2%	33.4%	58.1%	89.0%	66.5%	11.11
SmoothQuant-O2	75.0%	59.0%	79.2%	71.2%	33.0%	59.6%	88.0%	66.4%	11.14
SmoothQuant-O3	74.6%	58.9%	79.7%	71.2%	33.4%	59.9%	90.0%	66.8%	11.17

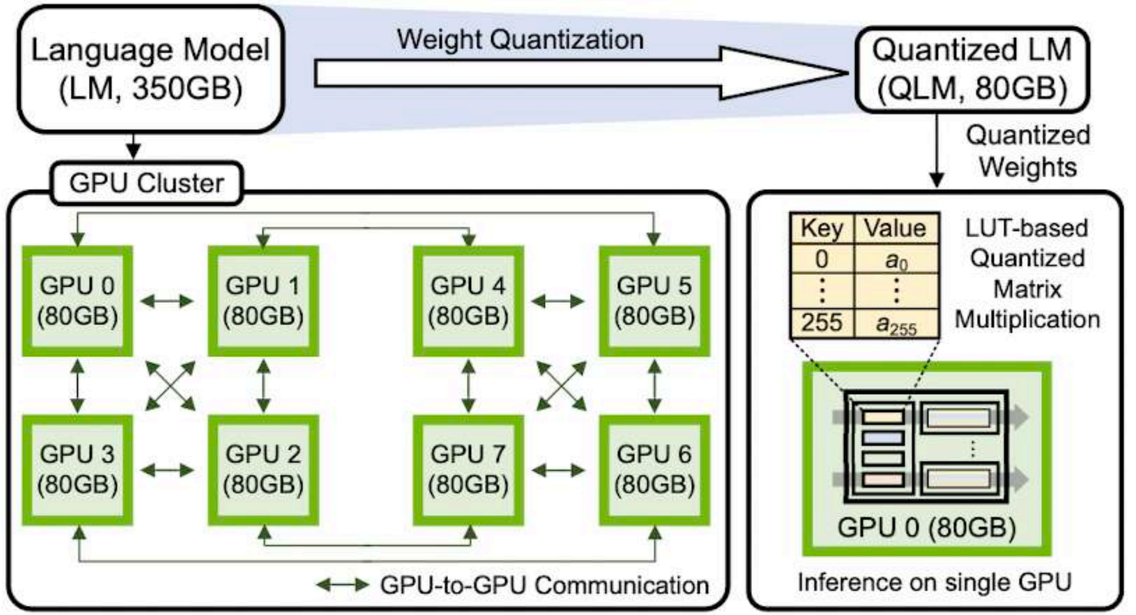


Figure 1: A large language model deployed with 8 GPUs supported by model parallelism, or with 1 GPU only using quantized weights. Assuming 80GB GPU memory size, 8 GPUs are required to serve a 350GB language model unless model compression is implemented.

Approaches

types

Post-Training Quantization	Quantization-Aware Training
• No data/~1000 calibration samples • ~1–10 GPU hours • **Calibrating** s, z • min–max, running min–max, MSE • No weights training • Useful for downstream	• Huge dataset (like for pretrain) • ~FT computational budget • ~100–1000 GPUs go brrr • **Training** s, z, weights using STE • Forget less -> better quality

wNa16

GPT-Q

Contribution summary

- **W4A16**
- 3,25x (**1,5x–2x** actually)
- **decoding only**
- Code and CUDA kernels published
- Highly popular in open source
- LLaMa.cpp CPU inference

OPT	13B	30B	66B	175B
Runtime	20.9m	44.9m	1.6h	4.2h
BLOOM	1.7B	3B	7.1B	176B
Runtime	2.9m	5.2m	10.0m	3.8h

Table 2: GPTQ runtime for full quantization of the 4 largest OPT and BLOOM models.

GPU	FP16	3bit	Speedup	GPU reduction
A6000 – 48GB	589ms	130ms	4.53×	8 → 2
A100 – 80GB	230ms	71ms	3.24×	5 → 1

Table 6: Average per-token latency (batch size 1) when generating sequences of length 128.

GPT-Q

Benchmarks

Method	Bits	OPT-175B Wiki2	PTB	C4	LAMB. ↑	BLOOM-176B Wiki2	PTB	C4	LAMB. ↑
Baseline	16	8.34	12.01	10.13	75.59	8.11	14.59	11.71	67.40
RTN	4	10.54	14.22	11.61	71.34	8.37	15.00	12.04	66.70
GPTQ	4	**8.37**	**12.26**	**10.28**	**76.80**	**8.21**	**14.75**	**11.81**	**67.71**
RTN	3	7.3e3	8.0e3	4.6e3	0	571.	107.	598.	0.17
GPTQ	3	**8.68**	**12.68**	**10.67**	**76.19**	**8.64**	**15.57**	**12.27**	**65.10**
GPTQ	3/g1024	8.45	12.48	10.47	77.39	8.35	15.01	11.98	67.47
GPTQ	3/g128	8.45	12.37	10.36	76.42	8.26	14.89	11.85	67.86

Table 5: Results summary for OPT-175B and BLOOM-176B. “g1024” and “g128” denote results with groupings of size 1024 and 128, respectively.

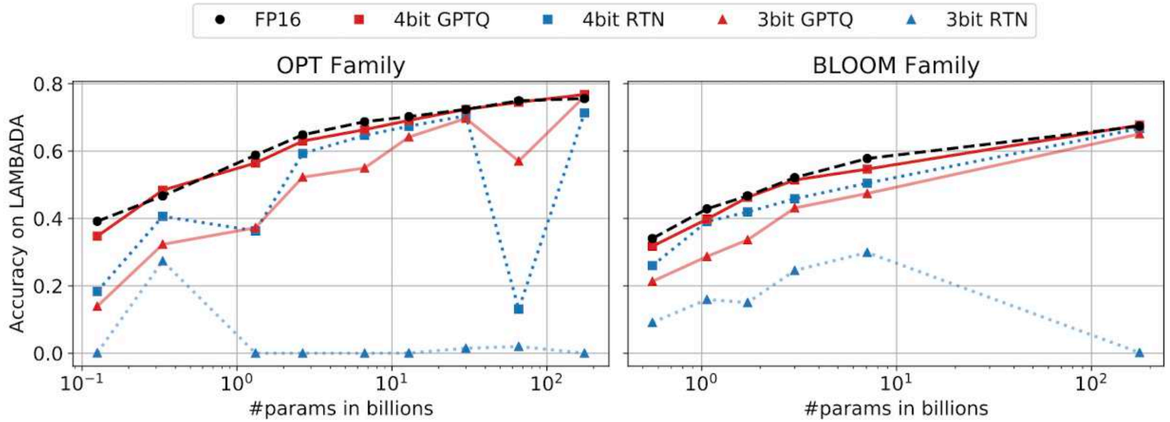


Figure 3: The accuracy of OPT and BLOOM models post-GPTQ, measured on LAMBADA.

Method	RN18 – 69.76 % 4bit	3bit	RN50 – 76.13% 4bit	3bit
AdaRound	69.34	68.37	75.84	75.14
AdaQuant	68.12	59.21	74.68	64.98
BRECQ	69.37	68.47	75.88	75.32
OBQ	69.56	68.69	75.72	75.24
GPTQ	69.37	67.88	75.71	74.87

Table 1: Comparison with state-of-the-art post-training methods for vision models.

OPT	Bits	125M	350M	1.3B	2.7B	6.7B	13B	30B	66B	175B
full	16	27.65	22.00	14.63	12.47	10.86	10.13	9.56	9.34	8.34
RTN	4	37.28	25.94	48.17	16.92	12.10	11.32	10.98	110	10.54
GPTQ	4	**31.12**	**24.24**	**15.47**	**12.87**	**11.39**	**10.31**	**9.63**	**9.55**	**8.37**
RTN	3	1.3e3	64.57	1.3e4	1.6e4	5.8e3	3.4e3	1.6e3	6.1e3	7.3e3
GPTQ	3	**53.85**	**33.79**	**20.97**	**16.88**	**14.86**	**11.61**	**10.27**	**14.16**	**8.68**

Table 3: OPT perplexity results on WikiText2.

BLOOM	Bits	560M	1.1B	1.7B	3B	7.1B	176B
full	16	22.42	17.69	15.39	13.48	11.37	8.11
RTN	4	25.90	22.00	16.97	14.76	12.10	8.37
GPTQ	4	**24.03**	**19.05**	**16.48**	**14.20**	**11.73**	**8.21**
RTN	3	57.08	50.19	63.59	39.36	17.38	571
GPTQ	3	**32.31**	**25.08**	**21.11**	**17.40**	**13.47**	**8.64**

Table 4: BLOOM perplexity results for WikiText2.

Challenges

Challenge

Quality drop

- Started from BERT
- Only weights — easy
- Activations — hard
- Outliers — the problem
- Isolated in some network locations, e.g. residual
- Needs engineering

Configuration	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	GLUE
FP32	57.27	93.12	88.36	89.09	89.72	84.91	91.58	70.40	**83.06**
W8A8	54.74	92.55	88.53	81.02	83.81	50.31	52.32	64.98	**71.03**
W32A8	56.70	92.43	86.98	82.87	84.70	52.80	52.44	53.07	**70.25**
W8A32	58.63	92.55	88.74	89.05	89.72	84.58	91.43	71.12	**83.23**

Table 1: Post-training quantization results on development sets of the GLUE benchmark (except WNLI). The metrics for these tasks can be found in the GLUE paper (Wang et al., 2018a); in all cases, higher is better. FP32 baseline is trained by the authors from the pre-trained checkpoint, see Appendix B.1 for details. We report a median over 5 runs with different random seeds.

Quantized activations	STS-B	MNLI	QNLI	RTE
none (FP32 model)	89.09	84.91	91.58	70.40
all	62.64	42.67	50.74	48.74
all, except softmax input	70.92	42.54	51.84	48.74
all, except sum of embeddings	67.57	46.82	51.22	51.26
all, except self-attention output	70.47	46.57	50.98	50.90
all, except softmax output	72.83	50.35	50.23	49.46
all, except residual connections after FFN	**81.57**	**82.56**	**89.73**	**67.15**
same as above, but for layers 10, 11 only	**79.40**	**81.24**	**88.03**	**63.90**

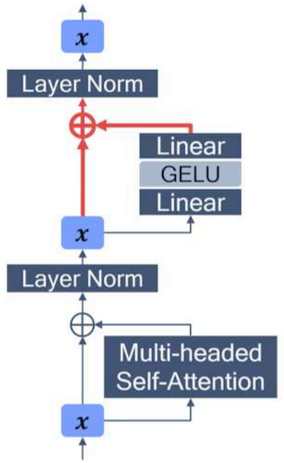
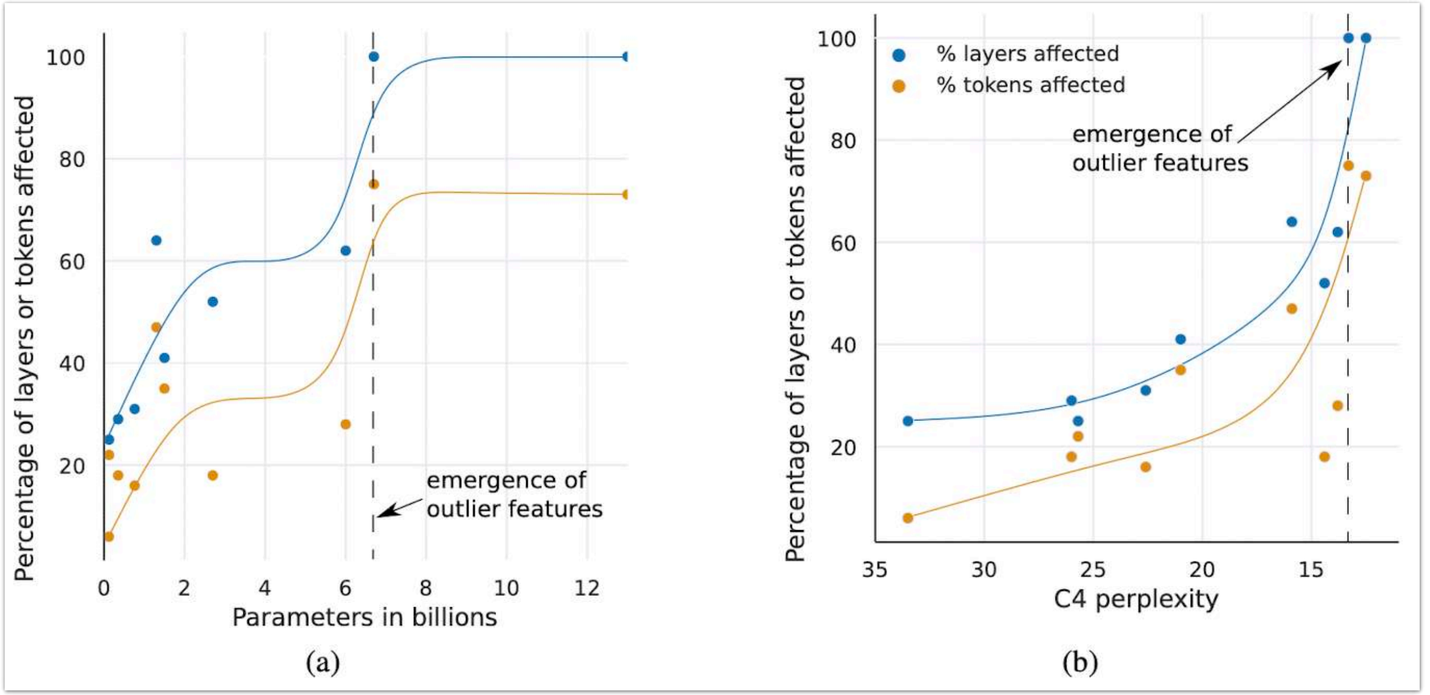


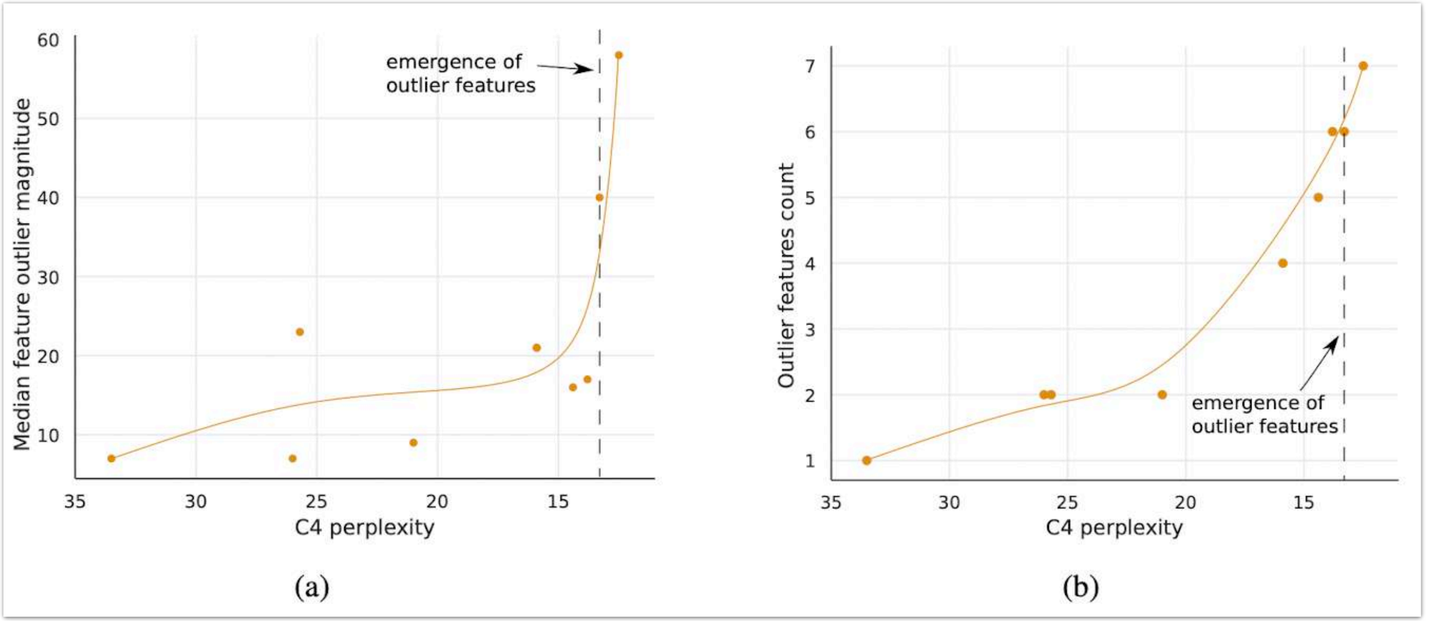
Table 2 & Figure 1: *Left*: Leave-one-out analysis for activation quantizers on problematic GLUE tasks. We set all weights to FP32 and use current min-max (with a batch size of 1) range estimator for activations. We report median score over 5 runs with different random seeds. *Right*: A schematic illustration of the attention layer in BERT. Hidden activation tensor is denoted by $\mathbf{x}$. $\oplus$ is an element-wise addition. A problematic residual connection sum after feed-forward network is highlighted in red.

LLM.int8()

Outliers

- Outliers occurs in QKVO projections, 1st FFN layer
- 6B->6,7B suddenly all layers with outliers
- Tokens/layers affected (exp) by decreasing perp





Impact

on top-1 softmax

Table 4: Summary statistics of outliers with a magnitude of at least 6 that occur in at least 25% of all layers and at least 6% of all sequence dimensions. We can see that the lower the C4 validation perplexity, the more outliers are present. Outliers are usually one-sided, and their quartiles with maximum range show that the outlier magnitude is 3-20x larger than the largest magnitude of other feature dimensions, which usually have a range of [-3.5, 3.5]. With increasing scale, outliers become more and more common in all layers of the transformer, and they occur in almost all sequence dimensions. A phase transition occurs at 6.7B parameters when the same outlier occurs in all layers in the same feature dimension for about 75% of all sequence dimensions (SDim). Despite only making up about 0.1% of all features, the outliers are essential for large softmax probabilities. The mean top-1 softmax probability shrinks by about 20% if outliers are removed. Because the outliers have mostly asymmetric distributions across the sequence dimension s, these outlier dimensions disrupt symmetric absmax quantization and favor asymmetric zeropoint quantization. This explains the results in our validation perplexity analysis. These observations appear to be universal as they occur for models trained in different software frameworks (fairseq, OpenAI, Tensorflow-mesh), and they occur in different inference frameworks (fairseq, Hugging Face Transformers). These outliers also appear robust to slight variations of the transformer architecture (rotary embeddings, embedding norm, residual scaling, different initializations).

			Outliers		Frequency			Top-1 softmax p	
Model	PPL↓	Params	Count	1-sided	Layers	SDims	Quartiles	w/ Outlier	No Outlier
GPT2	33.5	117M	1	1	25%	6%	(-8, -7, -6)	45%	19%
GPT2	26.0	345M	2	1	29%	18%	(6, 7, 8)	45%	19%
FSEQ	25.7	125M	2	2	25%	22%	(-40, -23, -11)	32%	24%
GPT2	22.6	762M	2	0	31%	16%	(-9, -6, 9)	41%	18%
GPT2	21.0	1.5B	2	1	41%	35%	(-11, -9, -7)	41%	25%
FSEQ	15.9	1.3B	4	3	64%	47%	(-33, -21, -11)	39%	15%
FSEQ	14.4	2.7B	5	5	52%	18%	(-25, -16, -9)	45%	13%
GPT-J	13.8	6.0B	6	6	62%	28%	(-21, -17, -14)	55%	10%
FSEQ	13.3	6.7B	6	6	100%	75%	(-44, -40, -35)	35%	13%
FSEQ	12.5	13B	7	6	100%	73%	(-63, -58, -45)	37%	16%

wNaM

FP8

FP8 Quantization: overview

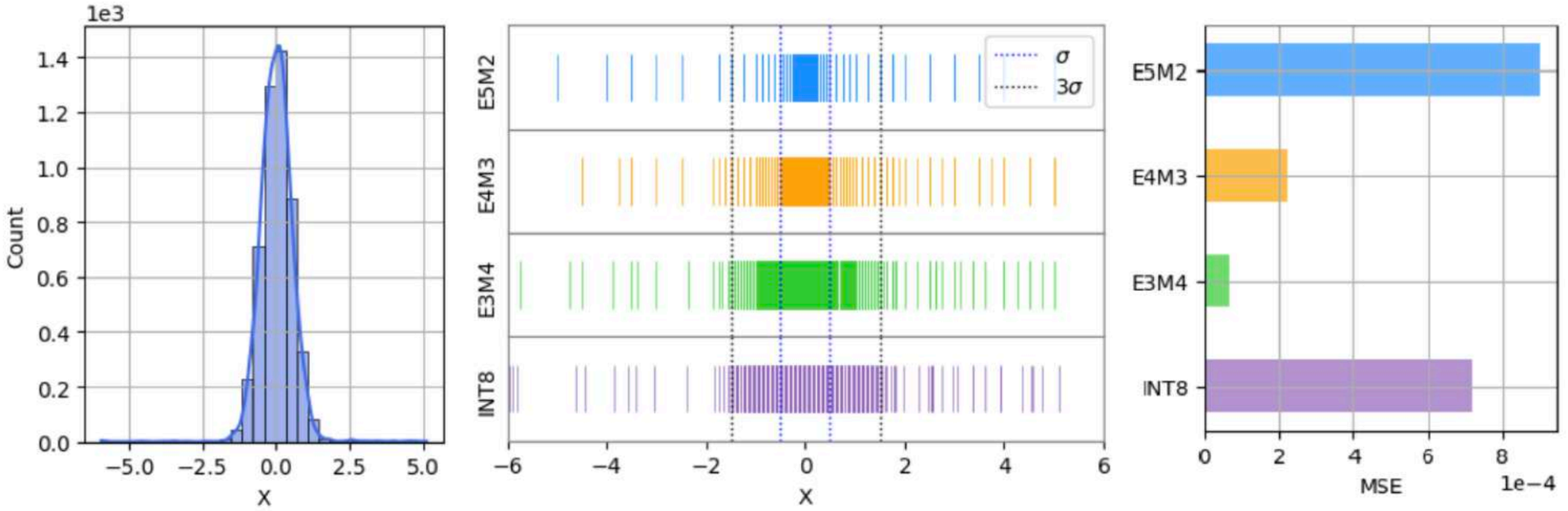


Figure 1. (**left**) Histogram of the tensor $X \sim \mathcal{N}(\mu = 0.0,\ \sigma^2 = 0.5)$, that contains a small number (1%) of outliers uniformly distributed between -6.0 to 6.0. (**center**) Distribution of quantized values represented by E5M2, E4M3, E3M4 and INT8 data formats. (**right**) Overall quantization error as measured by mean-square-error (MSE).

https://arxiv.org/pdf/2309.14592

FP8 Quantization: benefits

- Simple **static** amax-scaling quantization
- Several formats (E5M2, **E4M3**)
- **Per-tensor**/per-token quantization support
- Works better and faster than other quantizations, i.e. SmoothQuant
- KV-Cache - accurate and lightweight, **scale=1**

Model	Batch Size	Speedup (FP8 v.s. FP16)	Speedup (INT8 SQ v.s. FP16)
GPT-J	1	1.40x	1.40x
GPT-J	8	1.44x	1.30x
LLaMA-v2-7B	1	1.51x	1.47x
LLaMA-v2-7B	8	1.40x	1.32x

*The above benchmarks were run with Input Length=1024, Output Length=128, and TP=1 on H100 80GB.

Accuracy

Model	Quantization Methods	MMLU Baseline (FP16)	MMLU Post-quantization	MMLU Loss
Falcon-180B	FP8	70.4	70.3	0.14%
	INT8-SQ	70.4	68.6	2.56%
	INT4-AWQ	70.4	69.8	0.85%
Falcon-40B	FP8	56.1	55.6	0.89%
	INT8-SQ	56.1	54.7	2.50%
	INT4-AWQ	56.1	55.5	1.07%
LLaMA-v2-70B	FP8	69.1	68.5	0.87%
	INT8-SQ	69.1	67.2	2.75%
	INT4-AWQ	69.1	68.4	1.01%
MPT-30B	FP8	47.5	47.4	0.21%
	INT8-SQ	47.5	46.8	1.47%
	INT4-AWQ	47.5	46.5	2.11%

https://github.com/NVIDIA/TensorRT-LLM/blob/main/docs/source/blogs/quantization-in-TRT-LLM.md

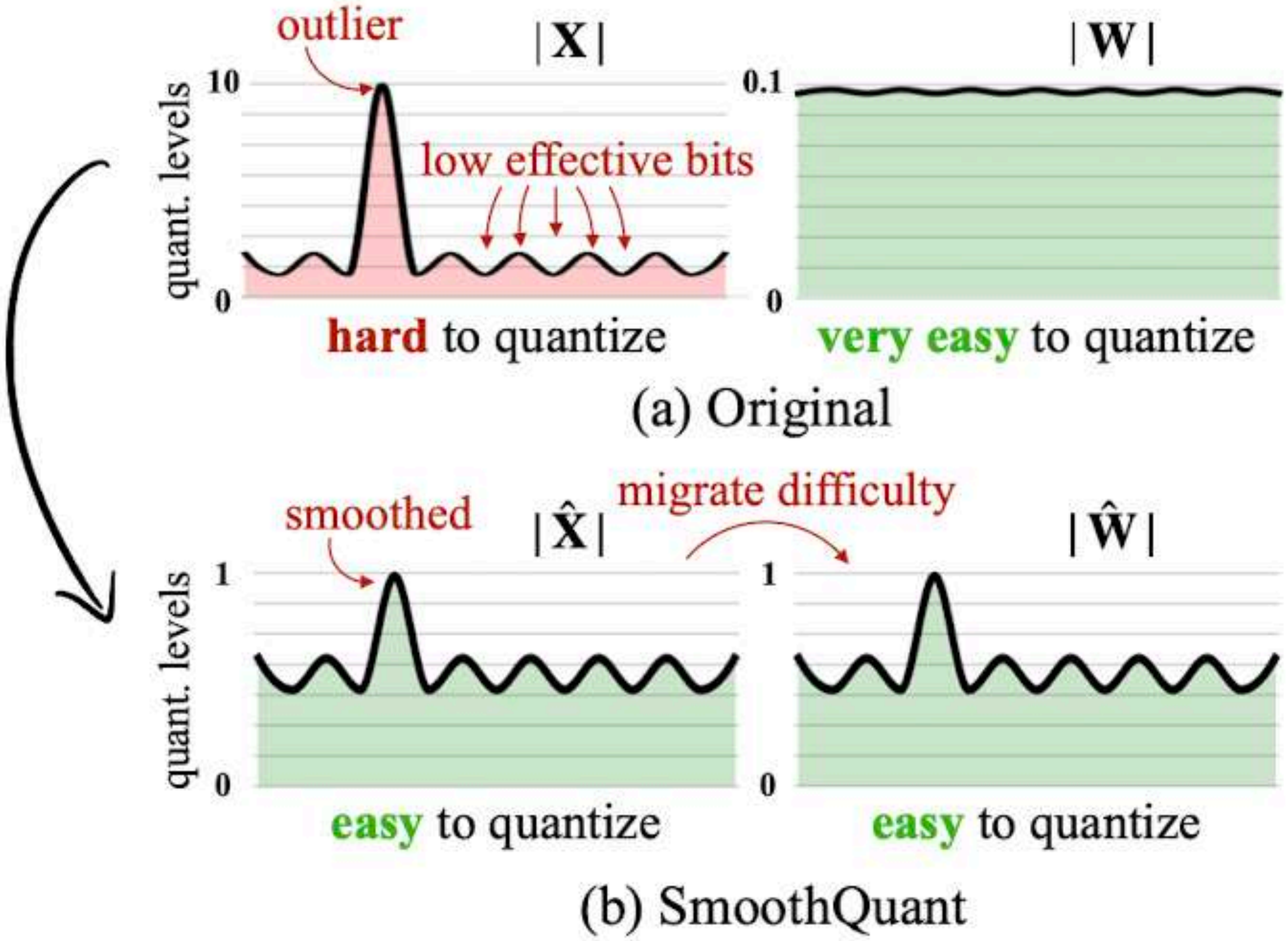
SmoothQuant

SmoothQuant

Intuition

- Hard to quant activations
- Migrate some difficulty to weights

	LLM (100B+) Accuracy	Hardware Efficiency
ZeroQuant	✗	✓
Outlier Suppression	✗	✓
`LLM.int8()`	✓	✗
SmoothQuant	✓	✓



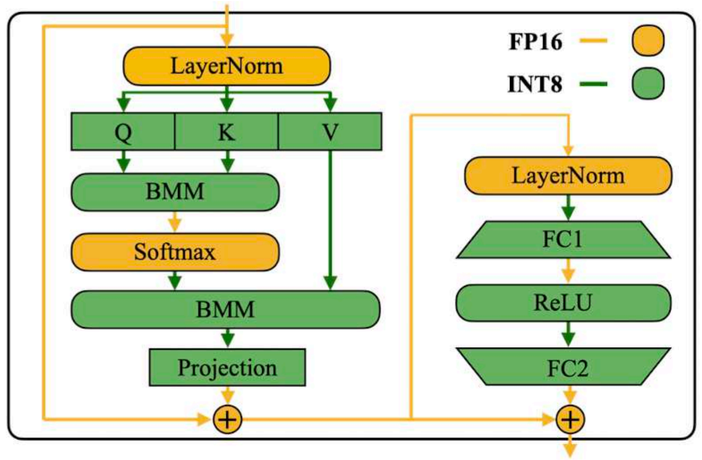
SmoothQuant

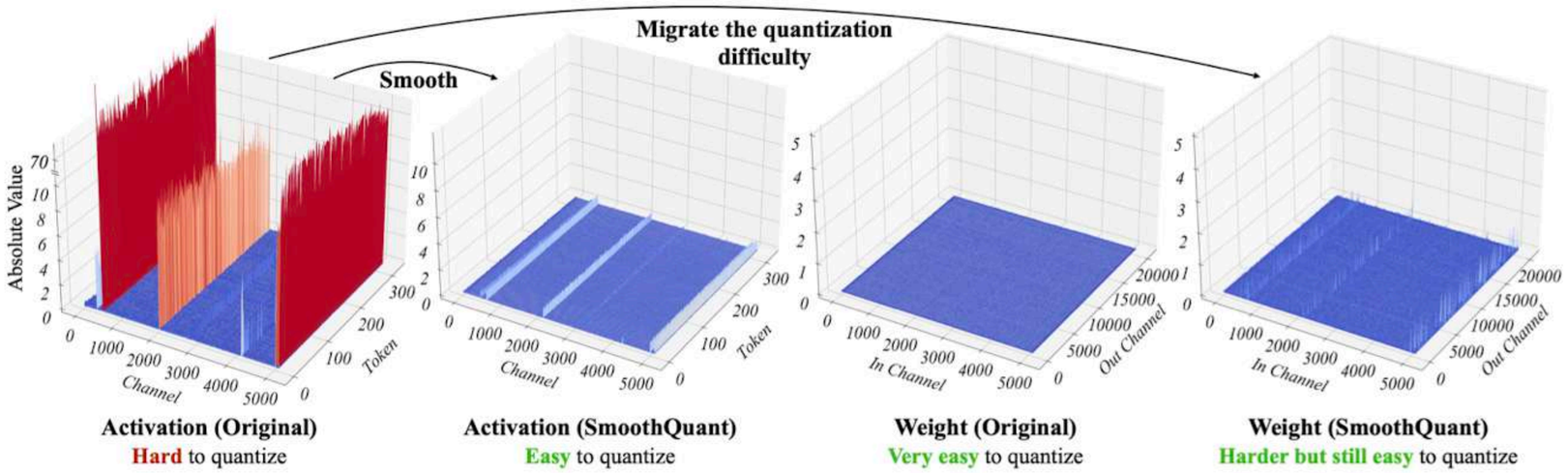
Motivation and method

- **W8A8**
- Per-channel scale is accurate but not efficient
- Apply to all BMM
- Fuse scaling with prev operations
- Good *s* ???

Table 2: Among different activation quantization schemes, only per-channel quantization (Bondarenko et al., 2021) preserves the accuracy, but it is *not* compatible (marked in gray) with INT8 `GEMM` kernels. We report the average accuracy on WinoGrande, HellaSwag, PIQA, and LAMBADA.

Model size (OPT-)	6.7B	13B	30B	66B	175B
FP16	64.9%	65.6%	67.9%	69.5%	71.6%
INT8 per-tensor	39.9%	33.0%	32.8%	33.1%	32.3%
INT8 per-token	42.5%	33.0%	33.1%	32.9%	31.7%
INT8 per-channel	64.8%	65.6%	68.0%	69.4%	71.4%





SmoothQuant

Scale tradeoff and choice

- 1 — hard W easy A
- 2 — easy W hard A
- 3 — ok W ok A
- Alpha grid search

1. $\mathbf{s}_j = \max(|\mathbf{X}_j|), j = 1, 2, ..., C_i,$
2. $\mathbf{s}_j = 1/\max(|\mathbf{W}_j|)$
3. $\mathbf{s}_j = \max(|\mathbf{X}_j|)^{\alpha}/\max(|\mathbf{W}_j|)^{1-\alpha}$

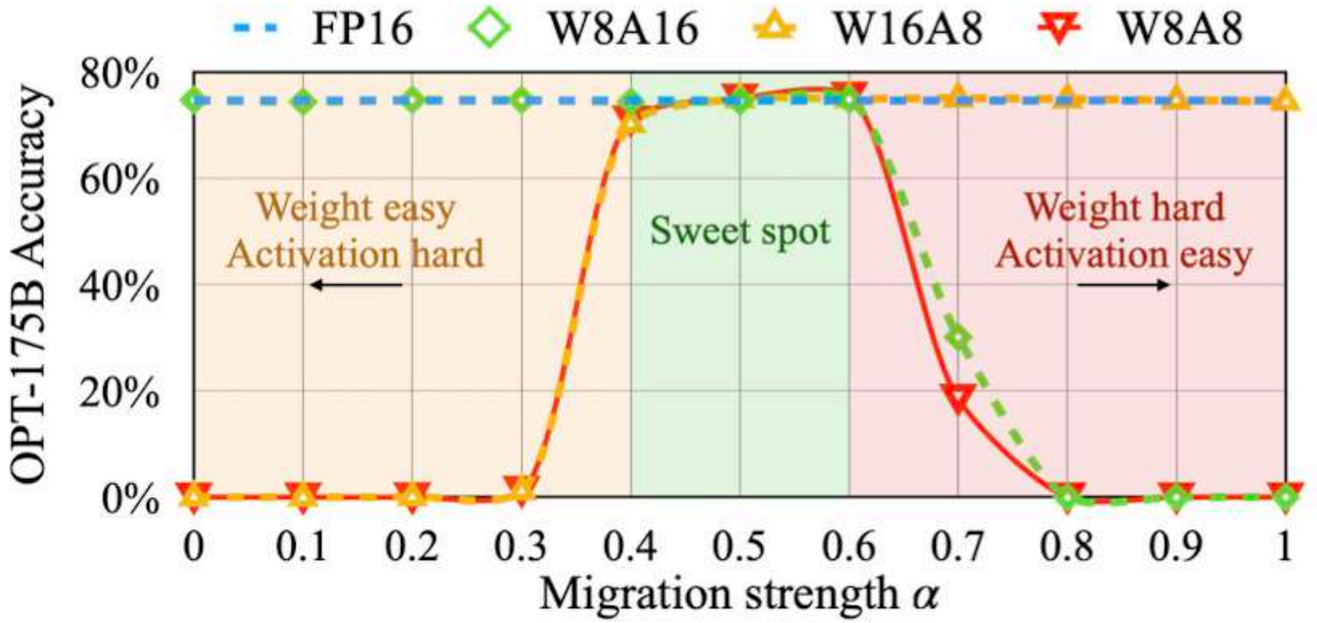
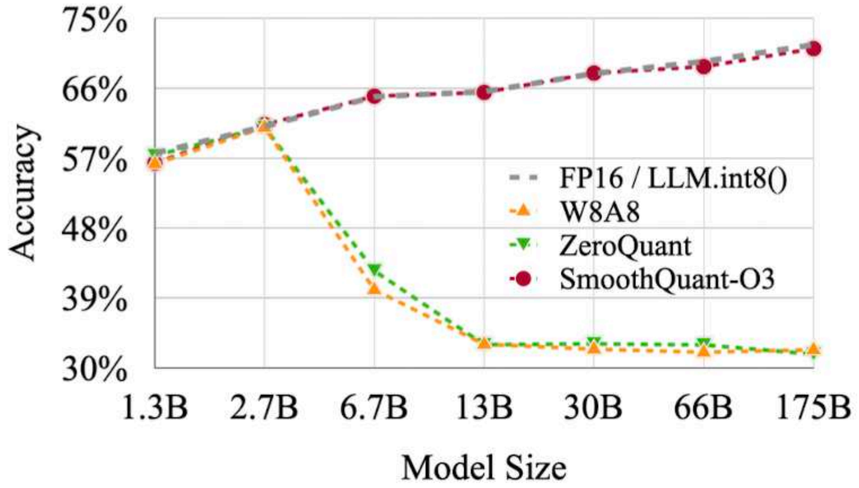


Figure 9: A suitable migration strength α (sweet spot) makes both activations and weights easy to quantize. If the α is too large, weights will be hard to quantize; if too small, activations will be hard to quantize.

SmoothQuant

Benchmarks

- O1–3 strategies
- ZS for OPT family
- Different families
- Preserves on all scales



Method	OPT-175B	BLOOM-176B	GLM-130B*
FP16	71.6%	68.2%	73.8%
W8A8	32.3%	64.2%	26.9%
ZeroQuant	31.7%	67.4%	26.7%
`LLM.int8()`	71.4%	68.0%	73.8%
Outlier Suppression	31.7%	54.1%	63.5%
SmoothQuant-O1	**71.2%**	68.3%	**73.7%**
SmoothQuant-O2	71.1%	**68.4%**	72.5%
SmoothQuant-O3	71.1%	67.4%	72.8%

Method	Weight	Activation
W8A8	per-tensor	per-tensor dynamic
ZeroQuant	group-wise	per-token dynamic
`LLM.int8()`	per-channel	per-token dynamic+FP16
Outlier Suppression	per-tensor	per-tensor static
SmoothQuant-O1	per-tensor	per-token dynamic
SmoothQuant-O2	per-tensor	per-tensor dynamic
SmoothQuant-O3	per-tensor	per-tensor static

OPT-175B	LAMBADA	HellaSwag	PIQA	WinoGrande	OpenBookQA	RTE	COPA	**Average↑**	**WikiText↓**
FP16	74.7%	59.3%	79.7%	72.6%	34.0%	59.9%	88.0%	66.9%	10.99
W8A8	0.0%	25.6%	53.4%	50.3%	14.0%	49.5%	56.0%	35.5%	93080
ZeroQuant	0.0%*	26.0%	51.7%	49.3%	17.8%	50.9%	55.0%	35.8%	84648
`LLM.int8()`	74.7%	59.2%	79.7%	72.1%	34.2%	60.3%	87.0%	66.7%	11.10
Outlier Suppression	0.00%	25.8%	52.5%	48.6%	16.6%	53.4%	55.0%	36.0%	96151
SmoothQuant-O1	74.7%	59.2%	79.7%	71.2%	33.4%	58.1%	89.0%	66.5%	11.11
SmoothQuant-O2	75.0%	59.0%	79.2%	71.2%	33.0%	59.6%	88.0%	66.4%	11.14
SmoothQuant-O3	74.6%	58.9%	79.7%	71.2%	33.4%	59.9%	90.0%	66.8%	11.17

SmoothQuant

Memory and latency speedups

Table 8: GPU Latency (ms) of different quantization schemes. The coarser the quantization scheme (from per-token to per-tensor, dynamic to static, O1 to O3, defined in Table 3), the lower the latency. SmoothQuant achieves lower latency compared to FP16 under all settings, while `LLM.int8()` is mostly slower. The batch size is 4.

Model	OPT-13B		OPT-30B	
Sequence Length	256	512	256	512
FP16	152.6	296.3	343.0	659.9
`LLM.int8()`	237.1	371.5	387.9	654.9
SmoothQuant-O1	124.5	243.3	246.7	490.7
SmoothQuant-O2	120.5	235.1	240.2	478.3
SmoothQuant-O3	112.1	223.1	227.6	458.4

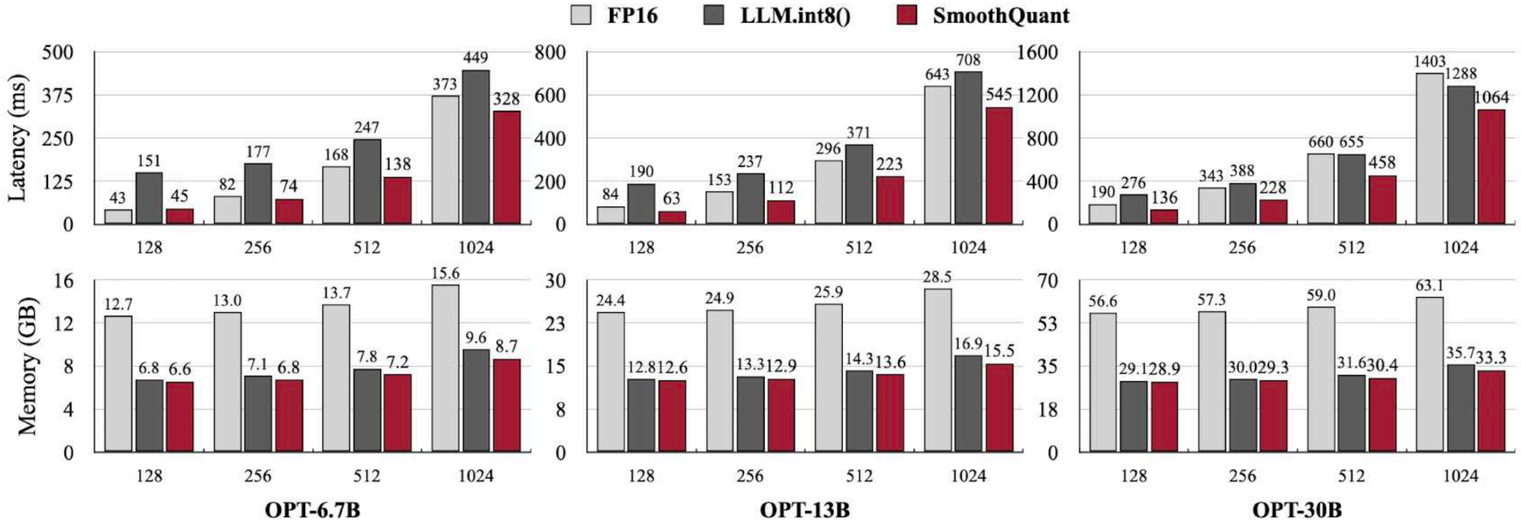


Figure 7: The PyTorch implementation of SmoothQuant-O3 achieves up to **1.51**× speedup and **1.96**× memory saving for OPT models on a single NVIDIA A100-80GB GPU, while `LLM.int8()` slows down the inference in most cases.

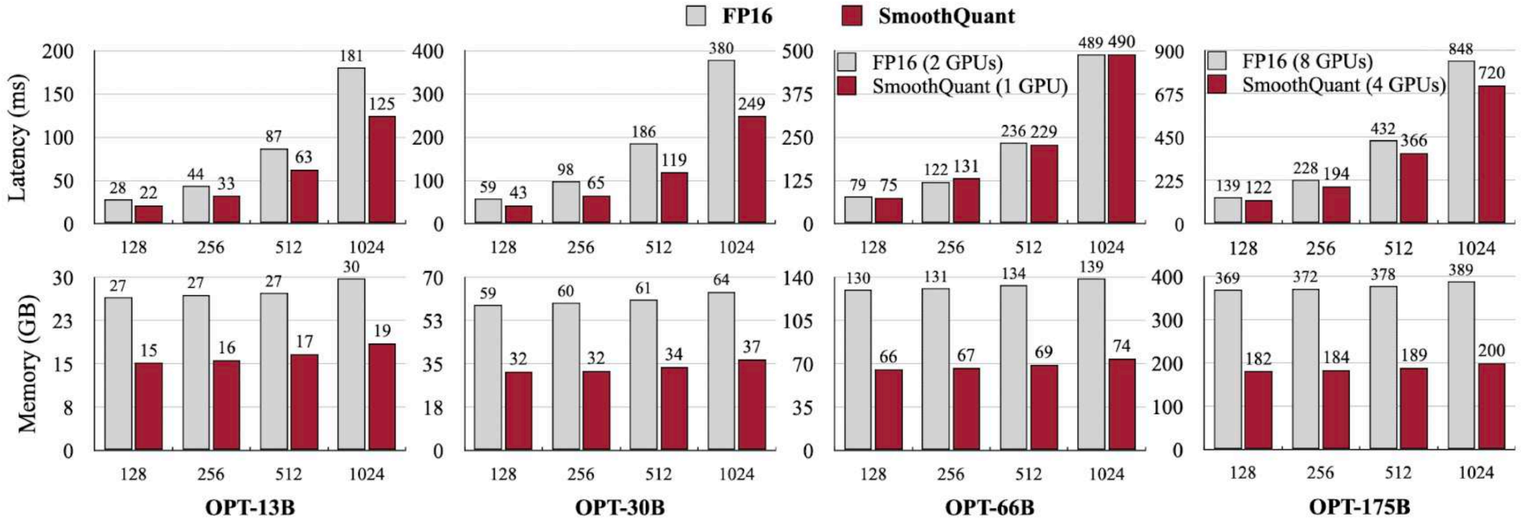


Figure 8: Inference latency (top) and memory usage (bottom) of the FasterTransformer implementation on NVIDIA A100-80GB GPUs. For smaller models, the latency can be significantly reduced with SmoothQuant-O3 by up to 1.56x compared to FP16. For the bigger models (OPT-66B and 175B), we can achieve similar or even faster inference using only **half** number of GPUs. Memory footprint is almost halved compared to FP16.

SpinQuant

SpinQuant

rotation idea

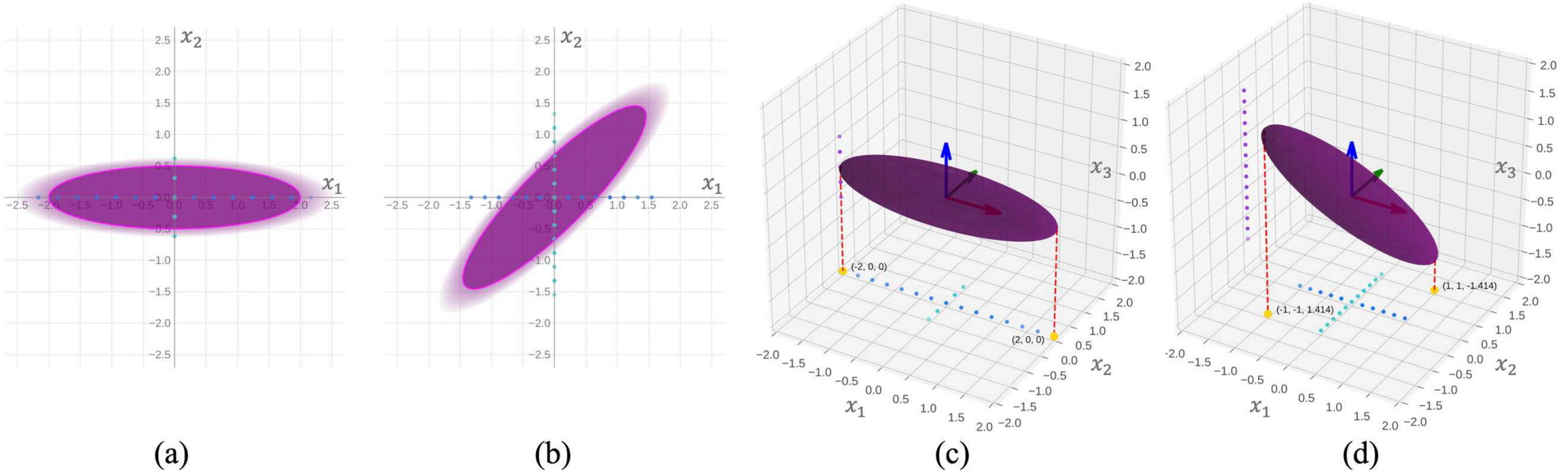


Figure 6: An illustration of how rotation helps reduce outliers and maximize quantization range utilization.

SpinQuant

rotation and outliers

- removes outliers
- stable & low kurtosis

The kurtosis is the fourth standardized moment, defined as

$$\operatorname{Kurt}[X] = \operatorname{E}\left[\left(\frac{X-\mu}{\sigma}\right)^4\right] = \frac{\operatorname{E}\left[(X-\mu)^4\right]}{\left(\operatorname{E}\left[(X-\mu)^2\right]\right)^2} = \frac{\mu_4}{\sigma^4},$$

- stable & low quant err

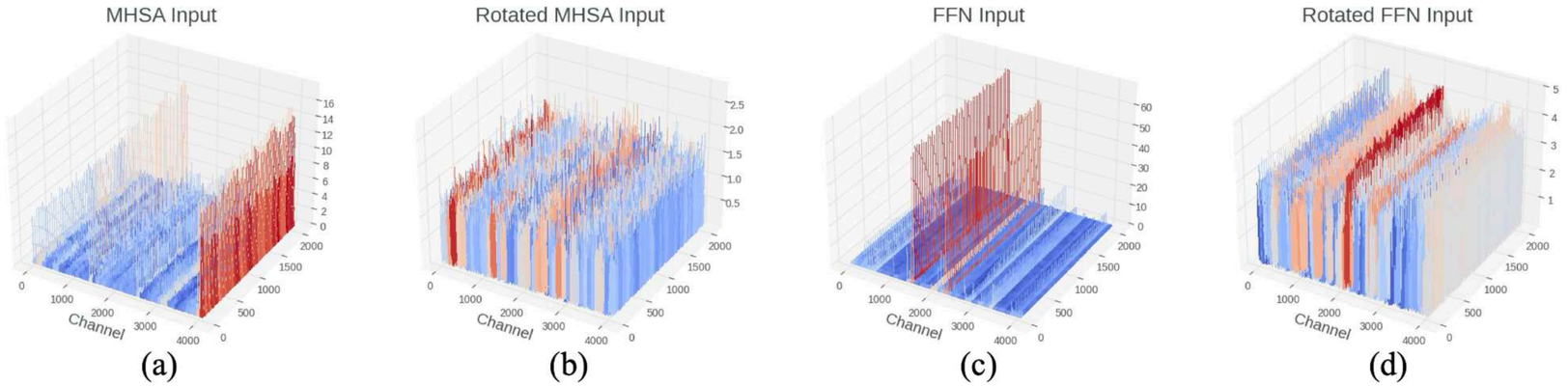


Figure 2: Activation distribution in LLaMA-2 7B model before and after rotation. Outliers exist in particular channels before rotation. Since channel-wise quantization is not supported in most hardware, outlier removal using rotation enables accurate token-wise or tensor-wise quantization.

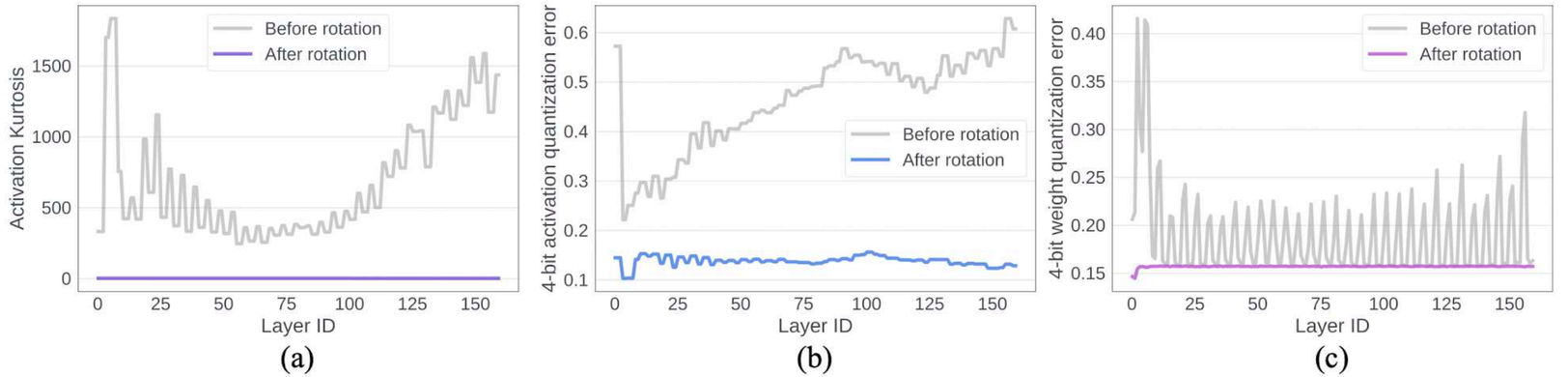


Figure 3: Outlier measurement and quantization error across input activation and weights in the five layers that take inputs from the residual (Q/K/V/Up/Gate-projection) of each block in the LLaMA-2 7B model. (a) After rotation, *kurtosis* of activation distributions is significantly reduced to approximately three across all layers. Quantization error is reduced after rotation in both (b) activations and (c) weights.

SpinQuant
method

- **w4a4kv4**
- smooth with rotation
- learnable rotation
- fuse rotation to W if possible
- else: Hadamard rotations

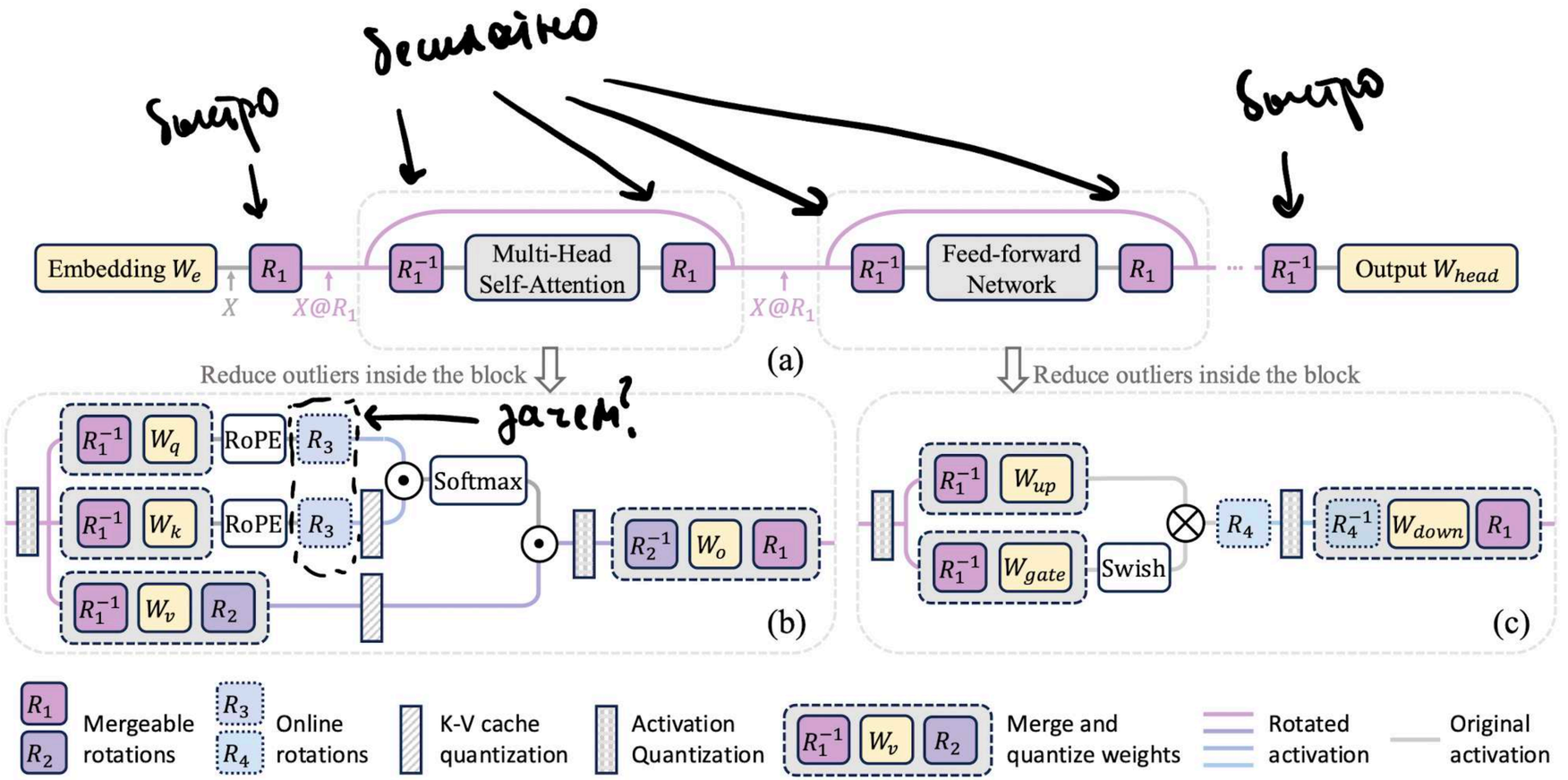


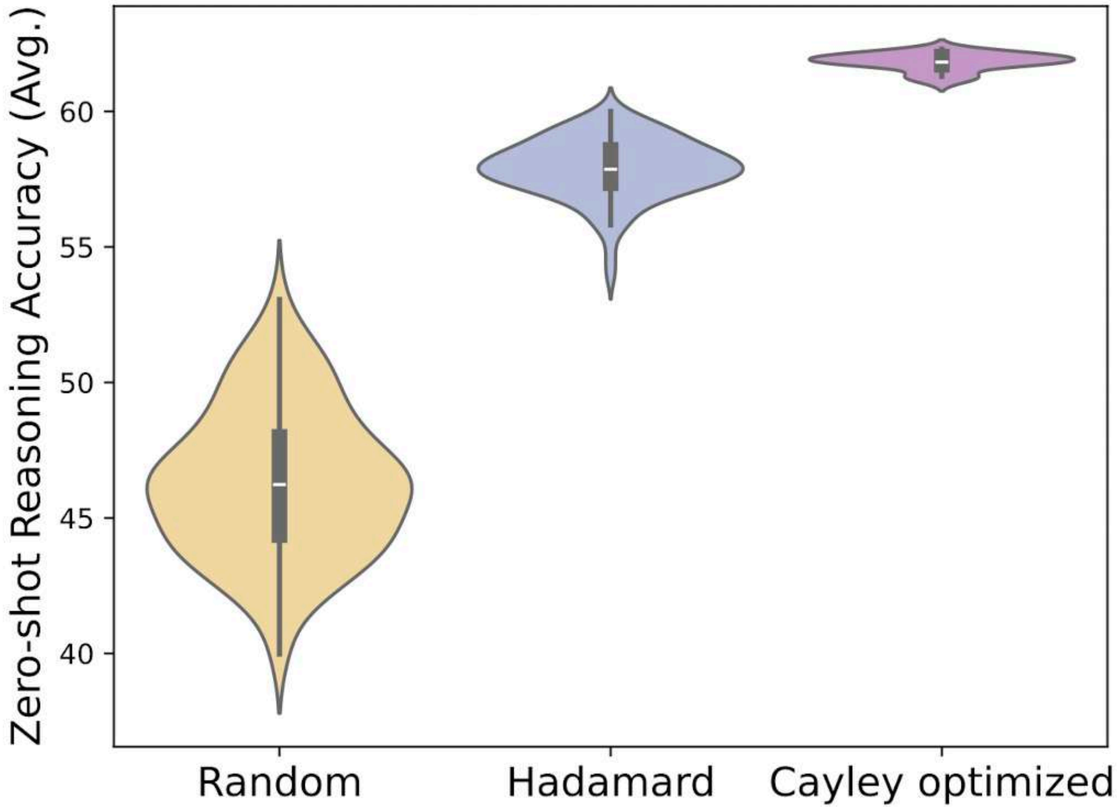
Figure 1: **Overall diagram of rotation.** (a) The residual stream can be rotated in the transformer network, resulting in numerically equivalent floating point networks before and after rotation. The rotated activations exhibit fewer outliers and are easier to quantize. (b) & (c) The rotation matrix can be integrated with the corresponding weight matrices and we further define R_2, R_3, and R_4 for reducing outliers inside the block.

SpinQuant

learnable rotations

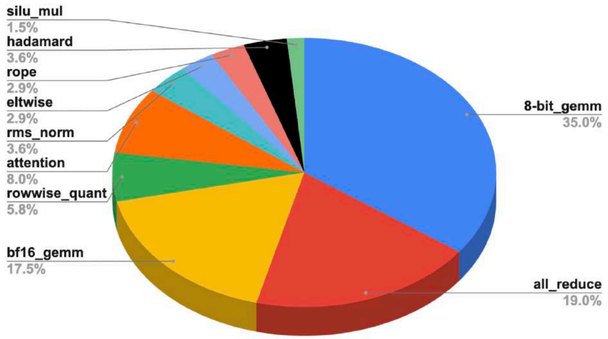
- init with Hadamard
- minimize quant err
- Cayley SGD
- better variance than random Hadamard

$$\arg\min_{R \in \mathcal{M}} \mathcal{L}_Q(R_1, R_2 \mid W, X)$$



SpinQuant
results

- ~3 pp loss on **w4a4kv4**
- best for 70B+
- great speedup



Module	Latency(ms)
8-bit gemm	0.048
all reduce	0.026
bf16 gemm	0.024
rowwise quant	0.008
attention	0.011
rms norm	0.005
eltwise	0.004
rope	0.004
hadamard	0.005
silu mul	0.002
total	0.137

Figure 7: The latency measurement of LLaMA-3 70B model with 8 H100 GPUs

Table 6: Real-time end-to-end speed measurement of LLaMA-3 8B on MacBook M1 Pro CPU.

Method	#Bits(W-A)	Decoding speed
FloatingPoint	16-16	177.15 ms/token
SpinQuant$_{no\ had}$	4-8	58.88 ms/token
SpinQuant$_{had}$	4-8	63.90 ms/token

kv size?

#Bits (W-A-KV)	Method	LLaMA-2 7B 0-shot[8] Avg.(↑)	LLaMA-2 7B Wiki (↓)	LLaMA-2 13B 0-shot[8] Avg.(↑)	LLaMA-2 13B Wiki (↓)	LLaMA-2 70B 0-shot[8] Avg.(↑)	LLaMA-2 70B Wiki (↓)	LLaMA-3.2 1B 0-shot[8] Avg.(↑)	LLaMA-3.2 1B Wiki (↓)	LLaMA-3.2 3B 0-shot[8] Avg.(↑)	LLaMA-3.2 3B Wiki (↓)	LLaMA-3 8B 0-shot[8] Avg.(↑)	LLaMA-3 8B Wiki (↓)	Mistral-7B 0-shot[8] Avg.(↑)	Mistral-7B Wiki (↓)
16-16-16	FloatingPoint	66.9	5.5	68.3	5.0	72.9	3.3	56.9	13.4	63.9	10.7	69.6	6.1	71.0	5.4
4-8-16	RTN	62.4	7.9	57.3	6.7	68.6	5.0	55.4	20.7	58.6	29.0	65.5	8.2	59.3	6.8
	SmoothQuant	58.9	7.5	63.6	6.1	70.6	4.1	47.1	1e2	55.6	3e2	61.0	10.7	–	–
	LLM-QAT	64.8	11.4	67.5	14.5	–	–	53.2	21.0	60.8	41.1	67.2	7.7	–	–
	AWQ (w4)	–	6.2	–	5.1	–	–	–	–	–	–	–	–	–	–
	OmniQuant (w4)	–	5.7	–	5.0	–	3.5	–	–	–	–	–	–	–	–
	QuIP# (w4)	–	**5.6**	–	5.0	–	**3.4**	–	–	–	–	–	–	–	–
	GPTQ	64.9	20.2	65.2	5.9	71.7	4.3	55.0	17.3	58.7	25.2	64.5	7.2	51.7	8.6
	SpinQuant$_{no\ had}$	**65.7**	5.8	**68.2**	5.1	72.1	3.7	56.0	15.3	61.4	11.6	**68.6**	6.7	68.8	5.7
	SpinQuant$_{had}$	**65.7**	5.7	68.1	**5.0**	**72.7**	3.5	**56.5**	**14.4**	**63.2**	**11.5**	68.4	**6.5**	**69.9**	**5.5**
4-8-8	RTN	62.5	7.9	57.6	6.7	68.4	5.0	55.7	20.7	58.4	28.8	65.3	8.2	58.9	6.7
	SmoothQuant	58.8	7.5	63.4	6.1	70.5	4.1	47.1	1e2	55.5	3e2	60.9	10.7	–	–
	LLM-QAT	64.6	11.4	67.5	14.2	–	–	53.1	21.0	60.5	39.3	66.9	7.6	–	–
	GPTQ	64.8	20.2	65.3	5.9	71.6	4.3	54.8	17.3	58.7	24.1	64.6	7.2	51.7	8.6
	SpinQuant$_{no\ had}$	**65.8**	5.8	68.1	**5.1**	72.2	3.7	55.7	15.3	61.8	11.7	68.6	6.7	69.4	5.7
	SpinQuant$_{had}$	**65.8**	**5.7**	**68.2**	**5.1**	**72.7**	**3.5**	**55.8**	**14.3**	**63.2**	**11.2**	**68.8**	**6.5**	**70.2**	**5.5**
4-4-16	RTN	35.6	2e3	35.3	7e3	35.1	2e5	41.2	1e2	42.1	7e2	43.9	2e2	41.4	4e2
	SmoothQuant	41.8	3e2	44.9	34.5	57.7	57.1	37.9	2e3	43.6	4e2	40.3	9e2	–	–
	LLM-QAT	47.8	12.9	34.3	4e3	–	–	42.0	62.1	46.9	37.6	44.9	42.9	–	–
	GPTQ	36.8	9e3	35.2	5e3	35.5	2e6	41.6	1e2	43.4	3e2	40.6	2e2	40.4	3e2
	SpinQuant$_{no\ had}$	57.0	9.2	61.8	7.2	61.0	7.3	44.8	48.4	52.9	22.4	51.9	18.6	52.7	13.4
	SpinQuant$_{had}$	**64.1**	**5.9**	**67.2**	**5.2**	**71.0**	**3.8**	**53.5**	**15.3**	**61.0**	**11.1**	**65.8**	**7.1**	**68.4**	**5.7**
4-4-4	RTN	37.1	2e3	35.5	7e3	35.0	2e5	40.6	2e2	41.2	8e2	43.1	3e2	41.4	4e2
	SmoothQuant	39.0	7e2	40.5	56.6	55.9	10.5	36.5	2e3	40.0	6e2	38.7	2e3	–	–
	LLM-QAT	44.9	14.9	35.0	4e3	–	–	41.5	76.2	45.9	42.0	43.2	52.5	–	–
	GPTQ	36.8	9e3	35.2	5e3	35.6	1e6	41.6	1e2	41.1	4e2	40.5	2e2	41.3	2e2
	SpinQuant$_{no\ had}$	56.0	9.2	60.7	7.1	62.0	7.4	45.3	47.7	52.9	22.4	52.6	18.6	52.4	13.7
	SpinQuant$_{had}$	**64.0**	**5.9**	**66.9**	**5.3**	**71.2**	**3.8**	**53.4**	**15.9**	**60.5**	**11.4**	**65.5**	**7.3**	**68.6**	**5.8**

Knowledge Distillation

Knowledge Distillation

General framework

- **Teacher** p(y|x): usually a LLM, e.g. GPT-3 (175B)
  - achieves SOTA quality
  - don’t fit inference computational budget
- **Student** q(y|x): small LM, e.g. T5 XL (3B)
  - unable to rich teacher’s quality by ordinary training
  - fits inference computational budget
- **Knowledge Distillation** (KD): process of teaching the student to imitate teacher’s performance

$$L(p(y|x), q_\theta(y|x)) \rightarrow \min_\theta$$

Hard-Label KD

Idea

- Sample targets from teacher
- SFT on that pairs
- Reproducing full sequences

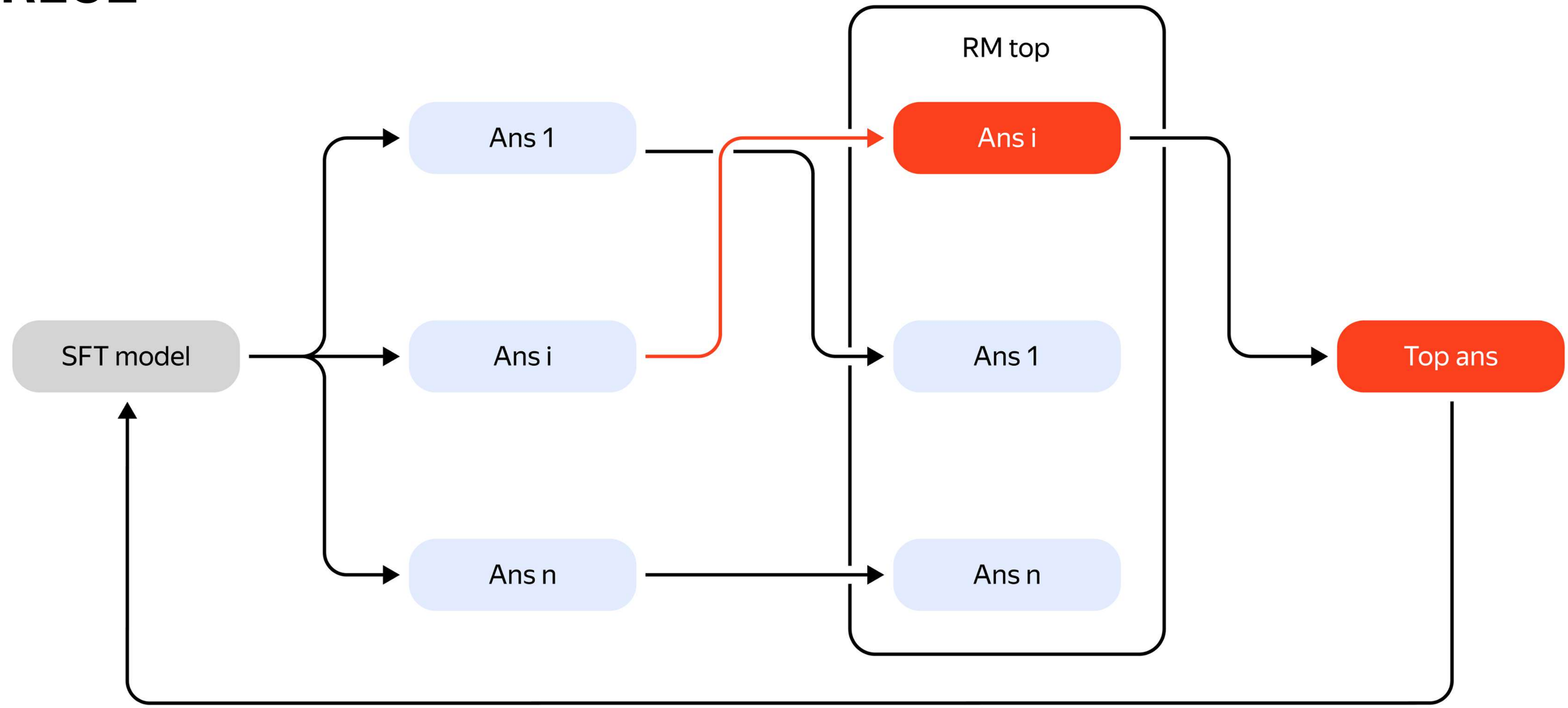
$$\mathop{\mathbb{E}}_{p(y)} \log q_\theta(y) \to \max_\theta$$

$$y^{(1)}, \ldots, y^{(N)} \sim p(y)$$

$$\frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \log q_\theta(y_t^{(n)} | y_{<t}^{(n)})$$

Hard-Label KD

RLCE



Soft-Label KD

Idea

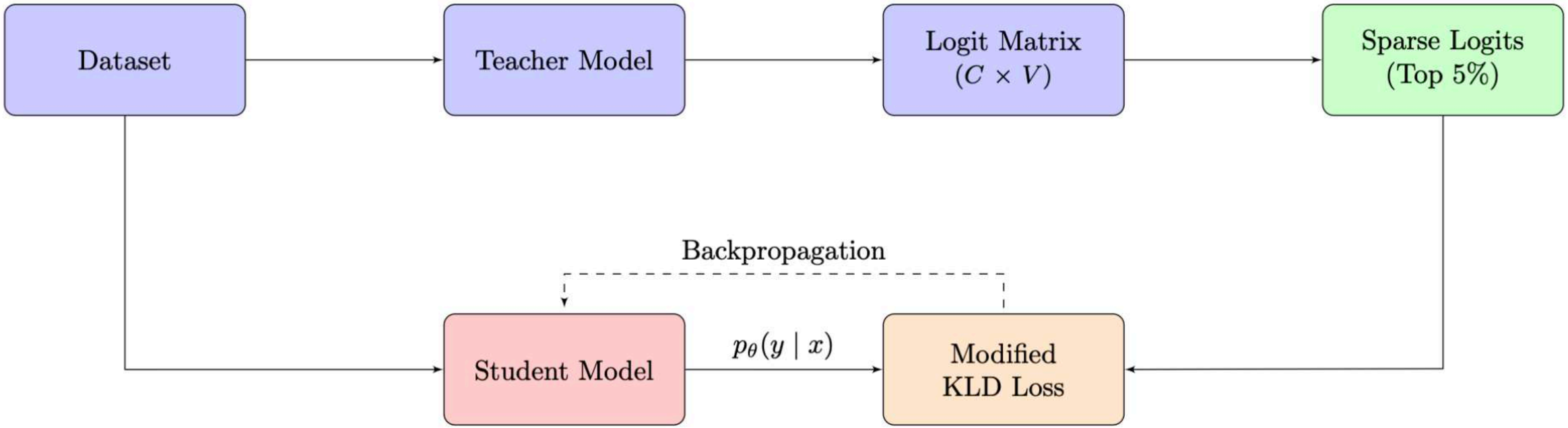
- Same as Hard-label, but reproduce logits also
- Sampling from teacher by default
- Dirty hack:
  - use targets from original dataset
  - compute logits in parallel

$$\frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\sum_{v\in\mathcal{V}} p(y_t^{(n)} = v|y_{<t}^{(n)}) \log q_\theta(y_t^{(n)} = v|y_{<t}^{(n)})$$

Soft-Label KD

SLIM

- 128k vocab size, too large to store even in distributed storage
- utilize top-5% logits for efficiency



KL KD

Definition and Idea

- “Distance” between distributions
- Monte-Carlo estimation

$$D_{\mathrm{KL}}(p(y) \,||\, q_\theta(y)) \to \min_\theta$$

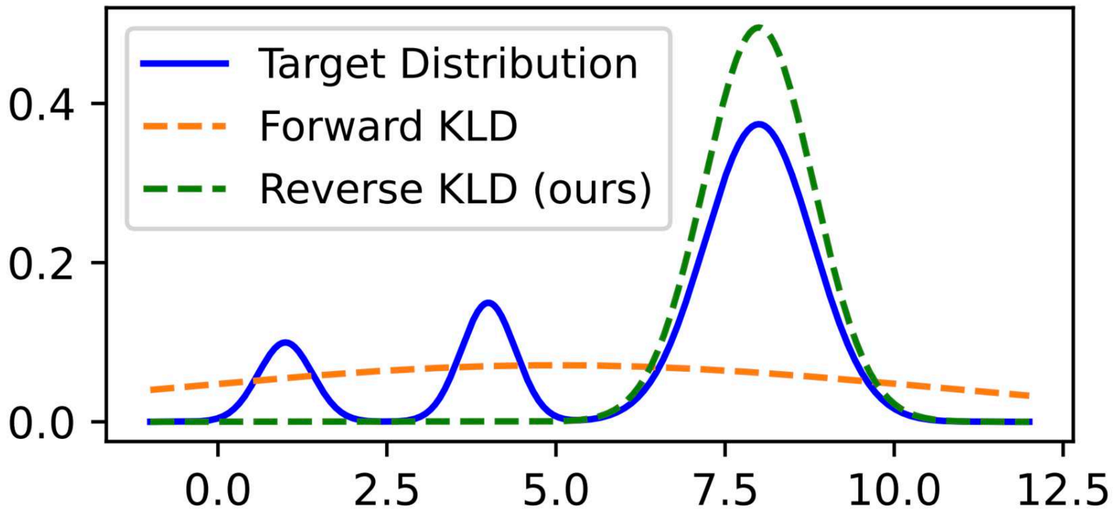
$$\mathop{\mathbb{E}}_{p(y)} \log \frac{p(y)}{q_\theta(y)} = -\mathop{\mathbb{E}}_{p(y)} \log q_\theta(y) + \mathrm{const} \to \min_\theta$$

KL KD

Issue

- Student's distribution should cover teacher's distribution
- While naturally student is less expressive than teacher

$$D_{\mathrm{KL}}(p(y) \,||\, q_\theta(y)) = \mathop{\mathbb{E}}_{p(y)} \log \frac{p(y)}{q_\theta(y)}$$



Reverse KLD

Solution

- Swap arguments of KL
- Student approx. only the top probs
- Entropy regularization

$$D_{\mathrm{KL}}(q_\theta(y) \,||\, p(y)) = \mathop{\mathbb{E}}_{q_\theta(y)} \log \frac{q_\theta(y)}{p(y)}$$

Reverse KLD

Solution

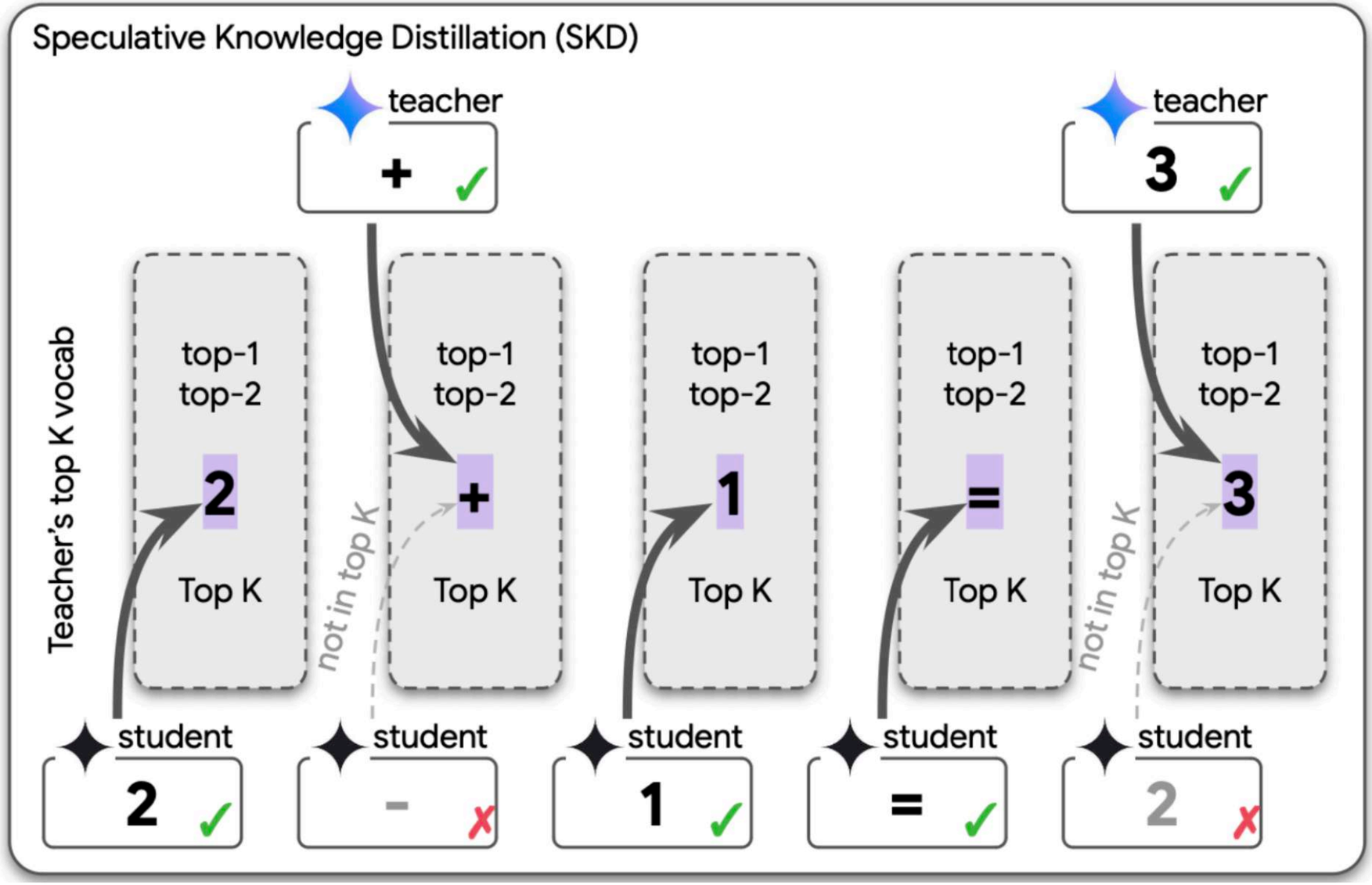
- Swap arguments of KL
- Student approx. only the top probs
- Entropy regularization
- Another problem occurs!
- Unable to differentiate by sampled y

$$D_{\mathrm{KL}}(q_\theta(y) \,||\, p(y)) = \mathop{\mathbb{E}}_{q_\theta(y)} \log \frac{q_\theta(y)}{p(y)}$$

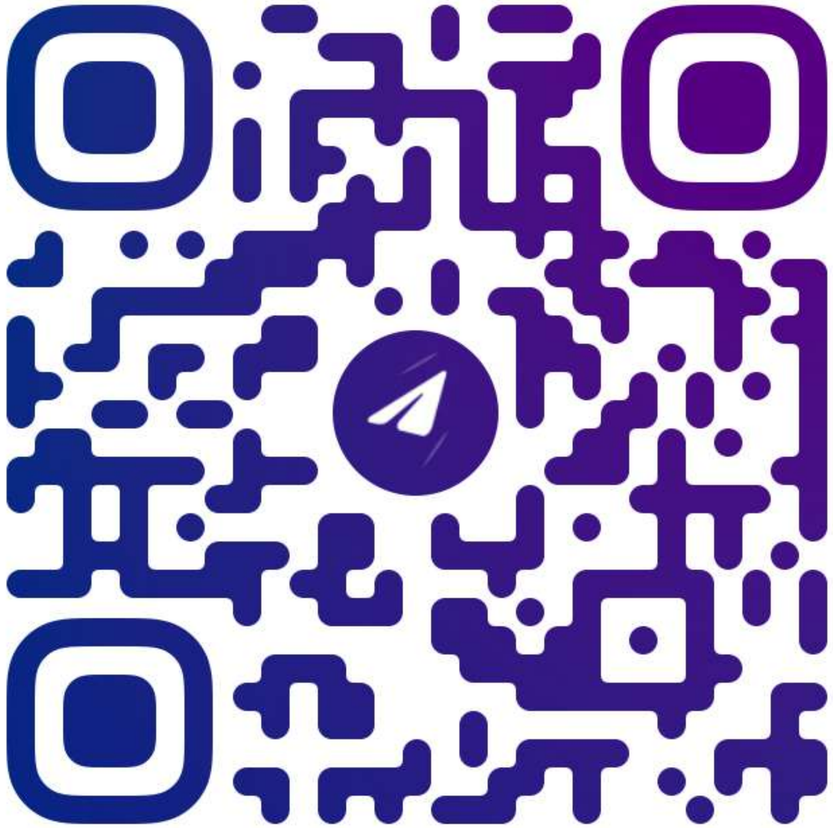
Speculative KD

simple approach

- student generates
- teacher monitors
- teacher's top-k
- no backprop through sampling



Thank you!



gorb-roman@yandex-team.ru YandexGPT