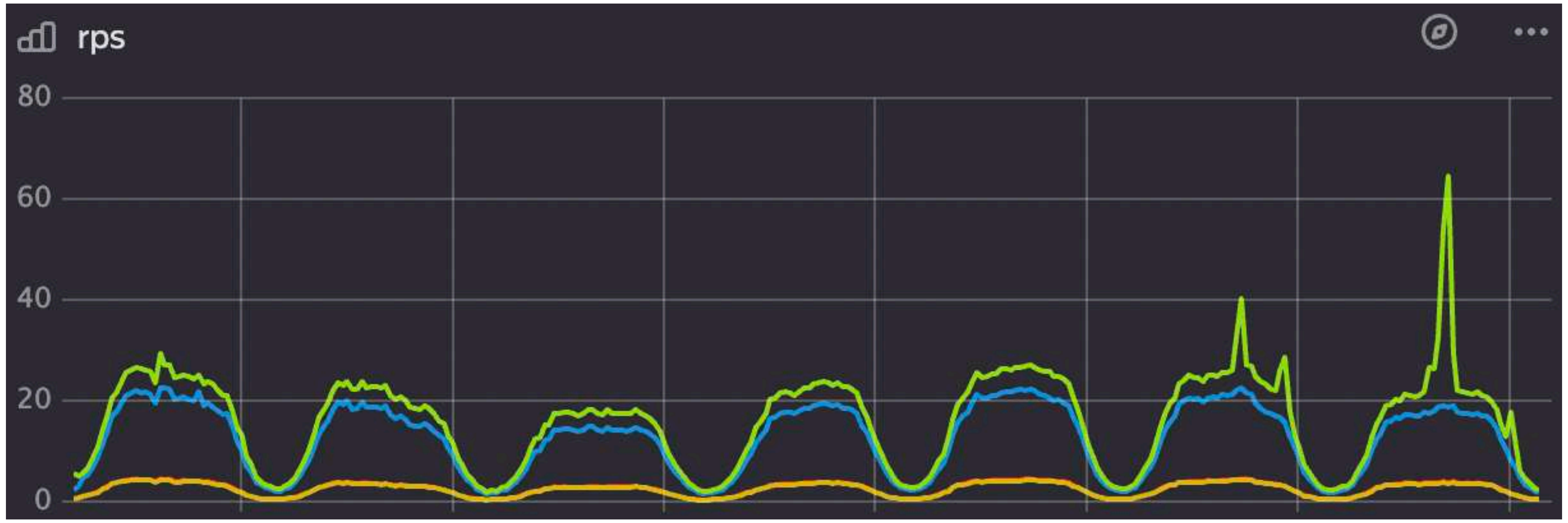
Efficient model inference

Efficient DL, Lecture IX

Smirnov Timofey
Based on materials of Roman Gorb from GPTWeek/Highload

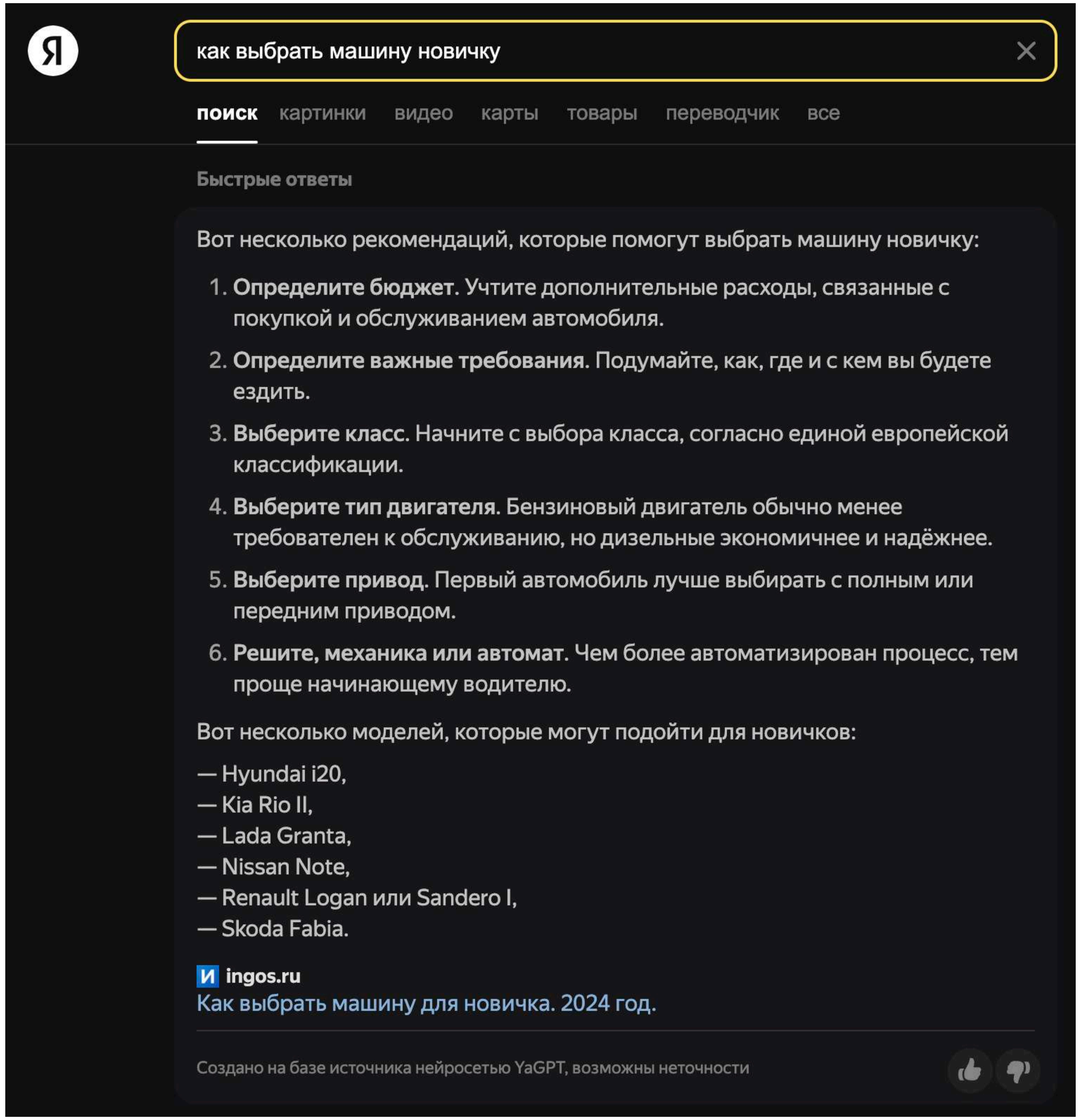
Пример
Метрики

- Предполагаемая нагрузка 1к RPS
- 20% cache-hit
- В пике может быть и больше
- 1GPU дает ~1rps при latency_q95 < 3s
- GPU дали 500



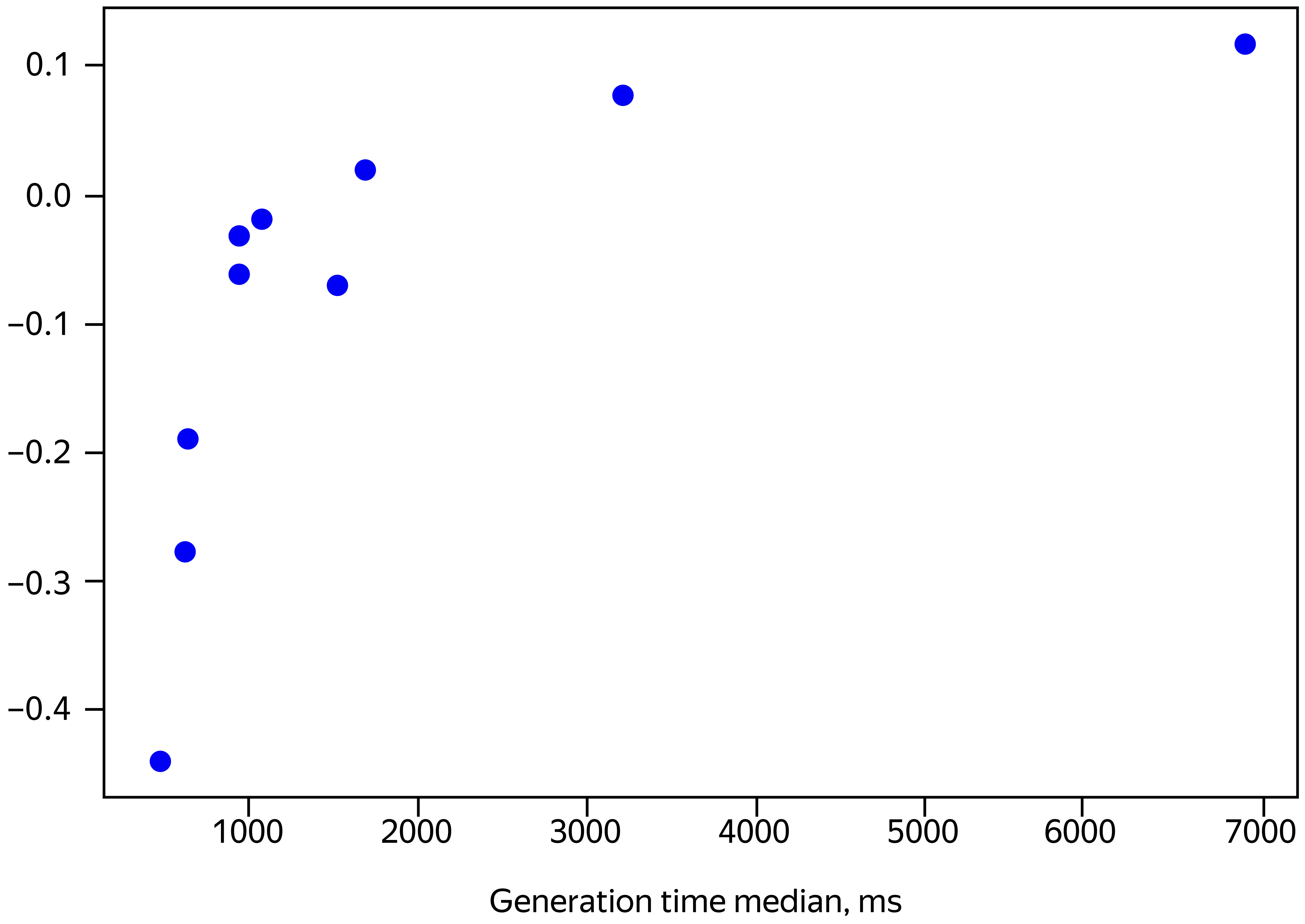
Пример

- Быстрый ответ
- Генеративный
- В среднем меньше секунды на генерацию
- Под капотом несколько генераций на запрос
- Относительно первой 7B модели ускорились в 30 раз

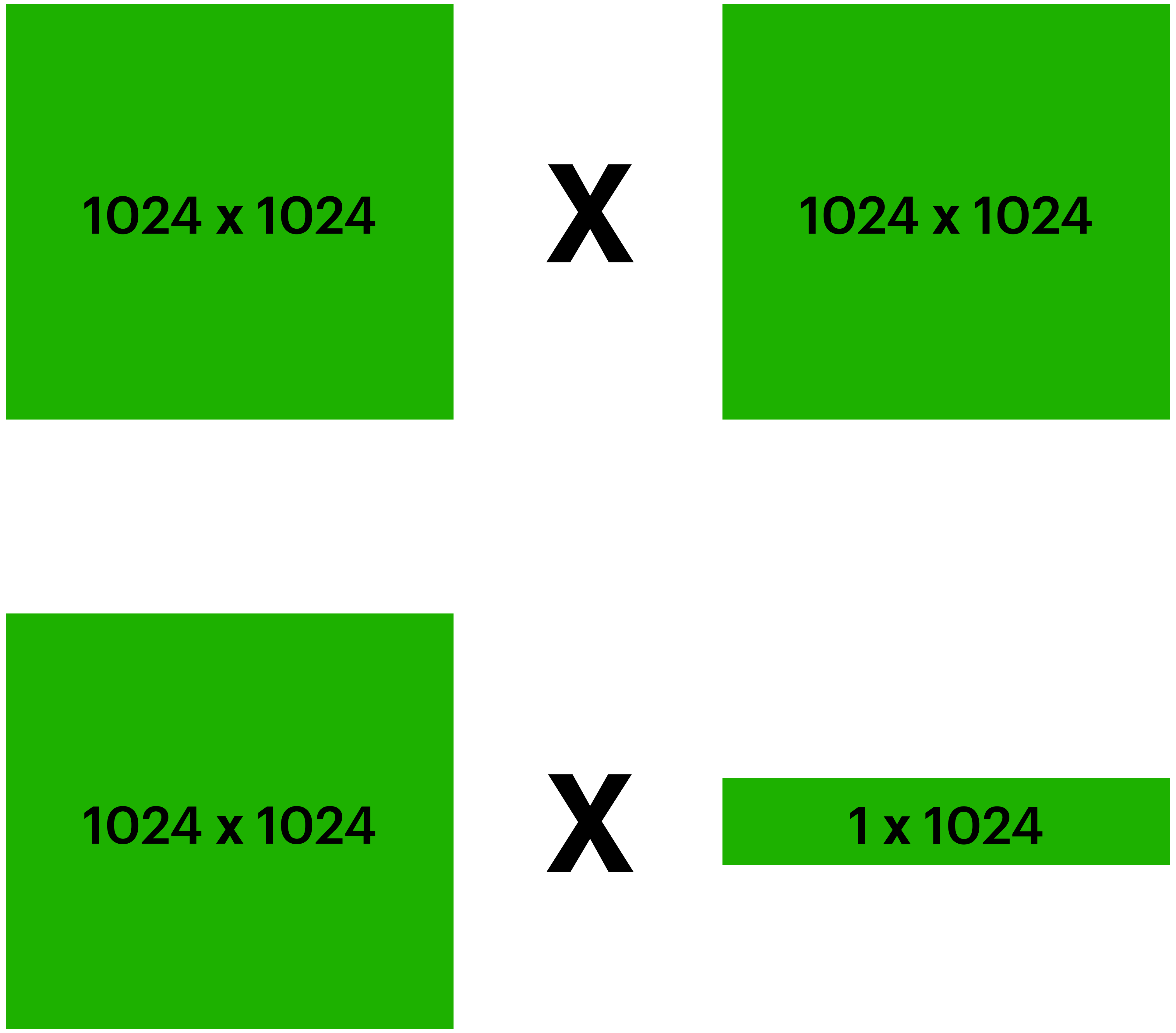


Pareto Curve
Quality vs Compute

- Latency/RPS по Оси X
- Качество по Оси Y



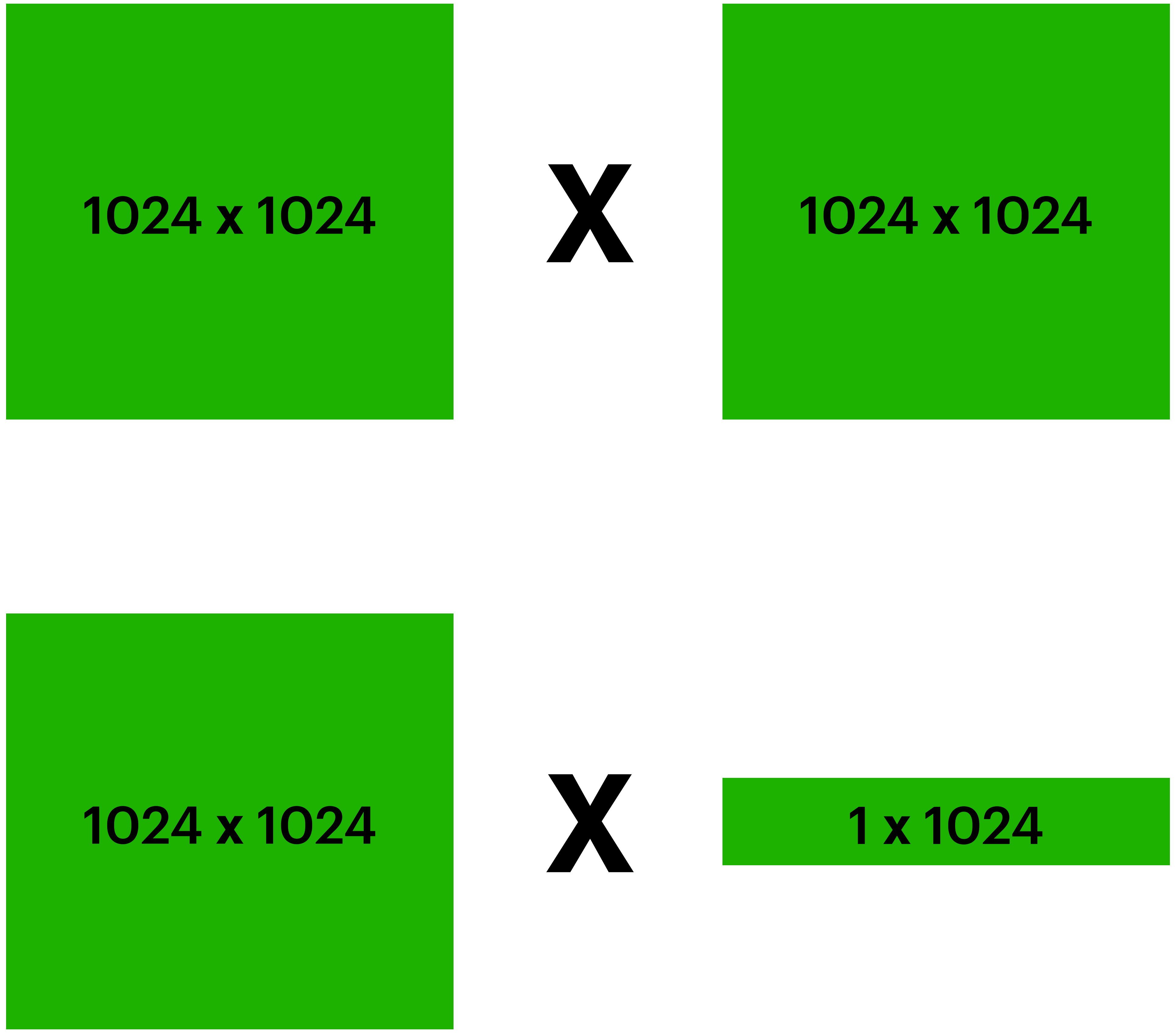
Compute bound vs Memory bound



Throughput = Y

Throughput?
1024*Y?

Compute bound vs Memory bound

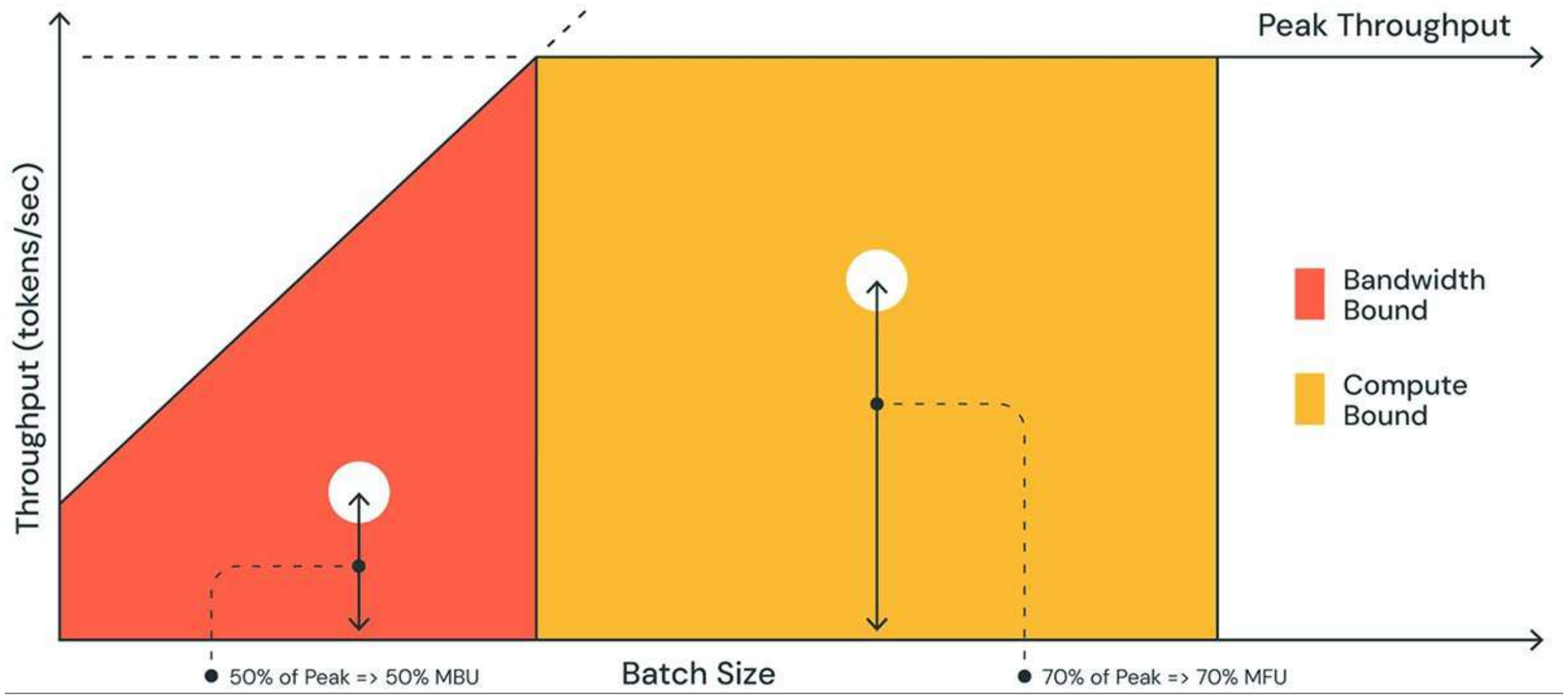


FLOP = 10^9
Bandwidth = 2 * 10^6

FLOP = 10^6
Bandwidth = 10^6

FLOPS vs Bandwidth utilisation

MBU = (achieved memory bandwidth) / (peak memory bandwidth)
achieved memory bandwidth = (total model parameter size + KV cache size) / TPOT



For example:
7B model in float16 has TPOT = 14ms
14 GB parameters in 14ms => 1TB/s bandwidth
peak bandwidth = 2TB / sec
MBU=50%

Distillation

Knowledge Distillation
General framework

- **Teacher p(y|x)**: usually a LLM, e.g. GPT-3 (175B) achieves SOTA quality
- **Student q(y|x)**: small LM, e.g. T5 XL (3B) unable to rich teacher's quality by ordinary training
- **Knowledge Distillation (KD)**: process of teaching the student to imitate teacher's performance

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

Hard-label Distillation

- Sample pairs from teacher
- Finetune model on that pairs

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

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

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

Soft-label Distillation

- Same as hard-label, but reproduce logits
- Sample from teacher by default
- Hack: use targets from original dataset and compute logits in parallel

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

KL Distillation

Idea

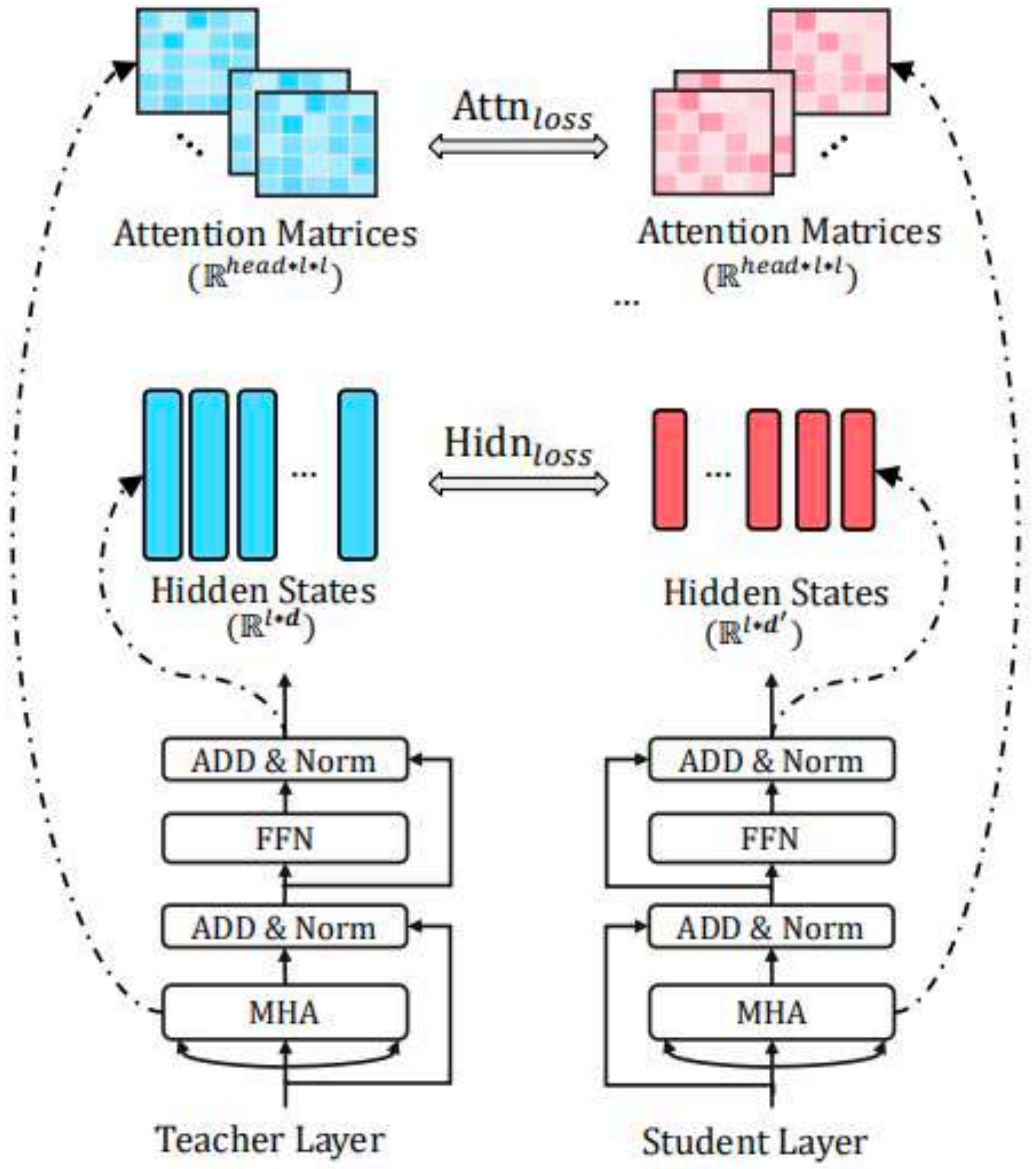
- Distance between distributions
- Monte-Carlo estimation

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

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

TinyBert

- $L_{attn} = \frac{1}{h} \sum MSE(A_i^S, A_i^T)$
- $L_{hidden} = MSE(H^S W_h, H^T)$
- $L_{embed} = MSE(E^S W_e, E^T)$
- $L_{pred} = CE(\frac{z^T}{t}, \frac{z^S}{t})$
- H^S, H^T - хиддены студента и учителя
- A^S, A^T - аттеншн мапы
- z - выходы модели

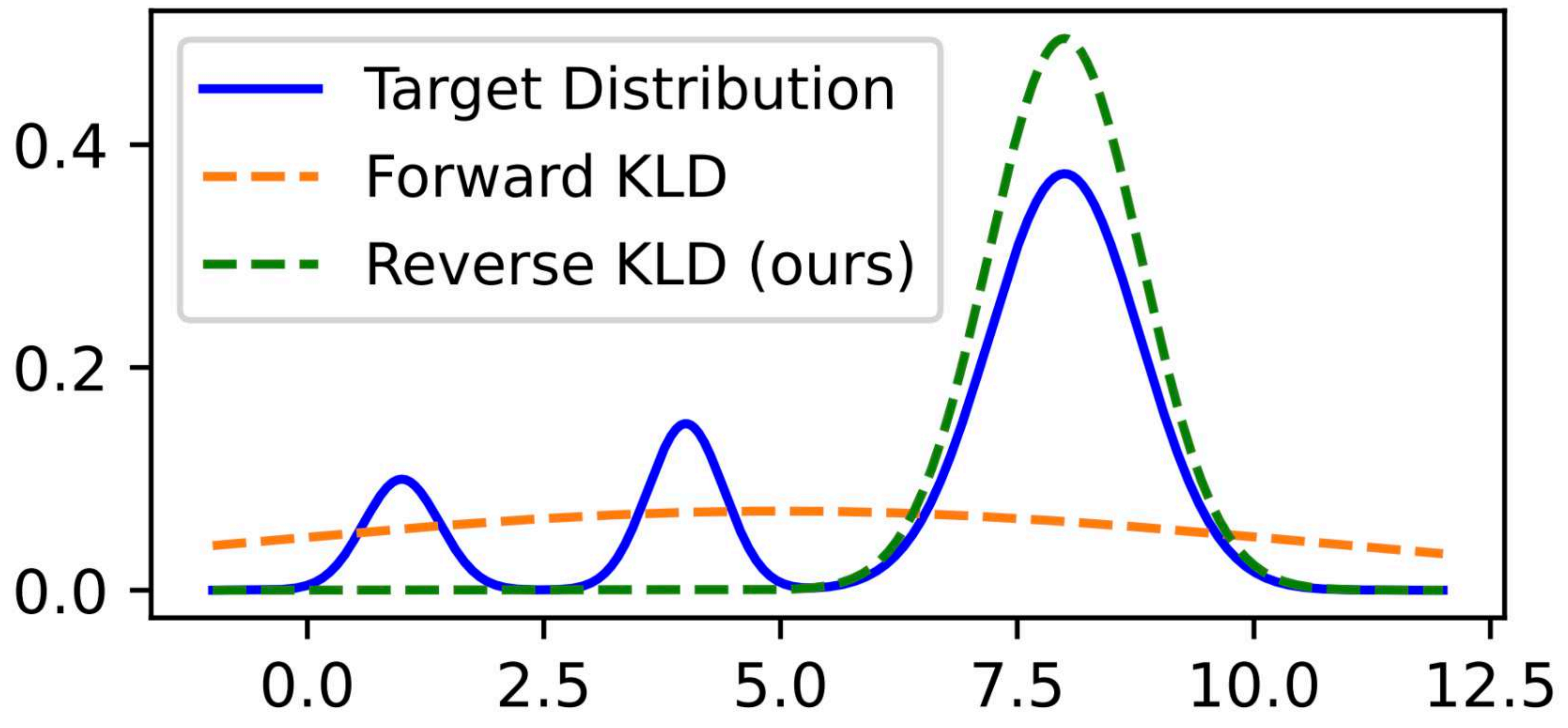


KL Distillation

LLM 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 KL
Solution

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

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

Reverse KL
Solution

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

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

Reverse KL

Solution

- Importance sampling
- Regularisation term
- Length normalisation

$$\tilde{q}_\theta(y_t \mid y_{<t}) = \alpha p(y_t \mid y_{<t}) + (1-\alpha) q_\theta(y_t \mid y_{<t})$$

$$R_{t+1}^{\text{Norm}} = \frac{1}{T-t-1} \sum_{k=t}^{T} \log \frac{q_\theta(y_k \mid y_{<k})}{p(y_k \mid y_{<k})}$$

$$\nabla_\theta \mathcal{L} = \mathop{\mathbb{E}}_{\tilde{q}_\theta(y)} \sum_{t=1}^{T} w_t \Bigg[R_{t+1}^{\text{Norm}} \nabla_\theta \log q_\theta(y_t \mid y_{<t}) + \nabla_\theta \mathop{\mathbb{E}}_{q_\theta(y_t \mid x, y_{<t})} \log \frac{q_\theta(y_t \mid y_{<t})}{p(y_t \mid y_{<t})} \Bigg]$$

SLIM
Alternative

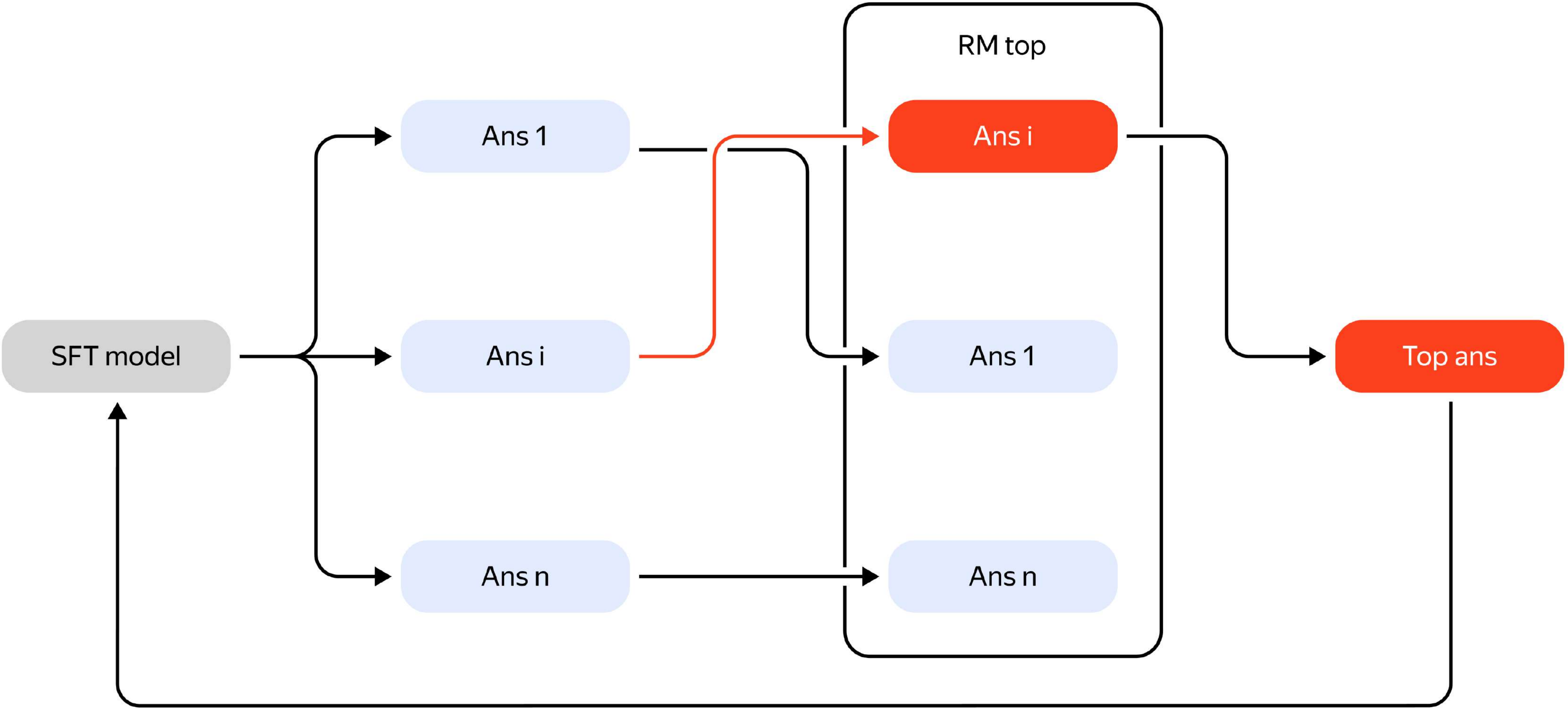
- Take only top %5 logits
- Compatibility with CERL(behaviour cloning)

$$L_{final} = L_{ce} + \alpha(1 - exp(-\frac{s_i}{t_i}))L_{kd}$$

	DollyEval		VicunaEval		SelfInst	
LLaMA	Rouge-L	GPT-4	Rouge-L	GPT-4	Rouge-L	GPT-4
Teacher (LLaMA 13B)	29.7	85.3	19.4	66.3	23.4	76.2
SFT	26.3	79.47	17.5	61.5	20.8	70.6
MiniLLM	28.7	82	19.8	63.8	20.5	72.1
SLIM (Ours)	**29.2**	**82.6**	**20.1**	**64.1**	**23.2**	**73.2**
LLaMA 2	Rouge-L	GPT-4	Rouge-L	GPT-4	Rouge-L	GPT-4
Teacher (LLaMA 2 13B)	30.2	88.9	21.3	69.3	25.1	79.1
SFT	26.5	80.3	18.3	62.8	21.3	73.4
SLIM (Ours)	**29.3**	**84.6**	**19.9**	**67.1**	**23.4**	**75.1**
MPT	Rouge-L	GPT-4	Rouge-L	GPT-4	Rouge-L	GPT-4
Teacher (MPT-30B-instruct)	44.0	94.7	19.3	67.3	23.5	76.1
SFT	28.6	79.5	16.62	60.7	19.9	70.7
SLIM (Ours)	**31.1**	**83.3**	**17.83**	**63.4**	**22.9**	**73.8**

Table 1: We report the Rouge-L and GPT-4 agreement scores on 3 different datasets across 3 different models. We do not have MiniLLM numbers for LLaMA 2 and MPT experiments since the authors did not open-source their models with these backbones.

CE RL



Наш опыт

- Hard label KD is all you need
- Если вбухать ОЧЕНЬ много компьюта, то даже супермаленькую сетку можно сделать очень умной
- Главное не забывать про diversity
- RL as distillation very good

Quantization

Квантизация
Формально

- FP16 -> INT 1/2/4/8
- VRAM and latency/ rps boost
- WxAy
- Symmetric vs Asymmetric
- Granularity
- Grid

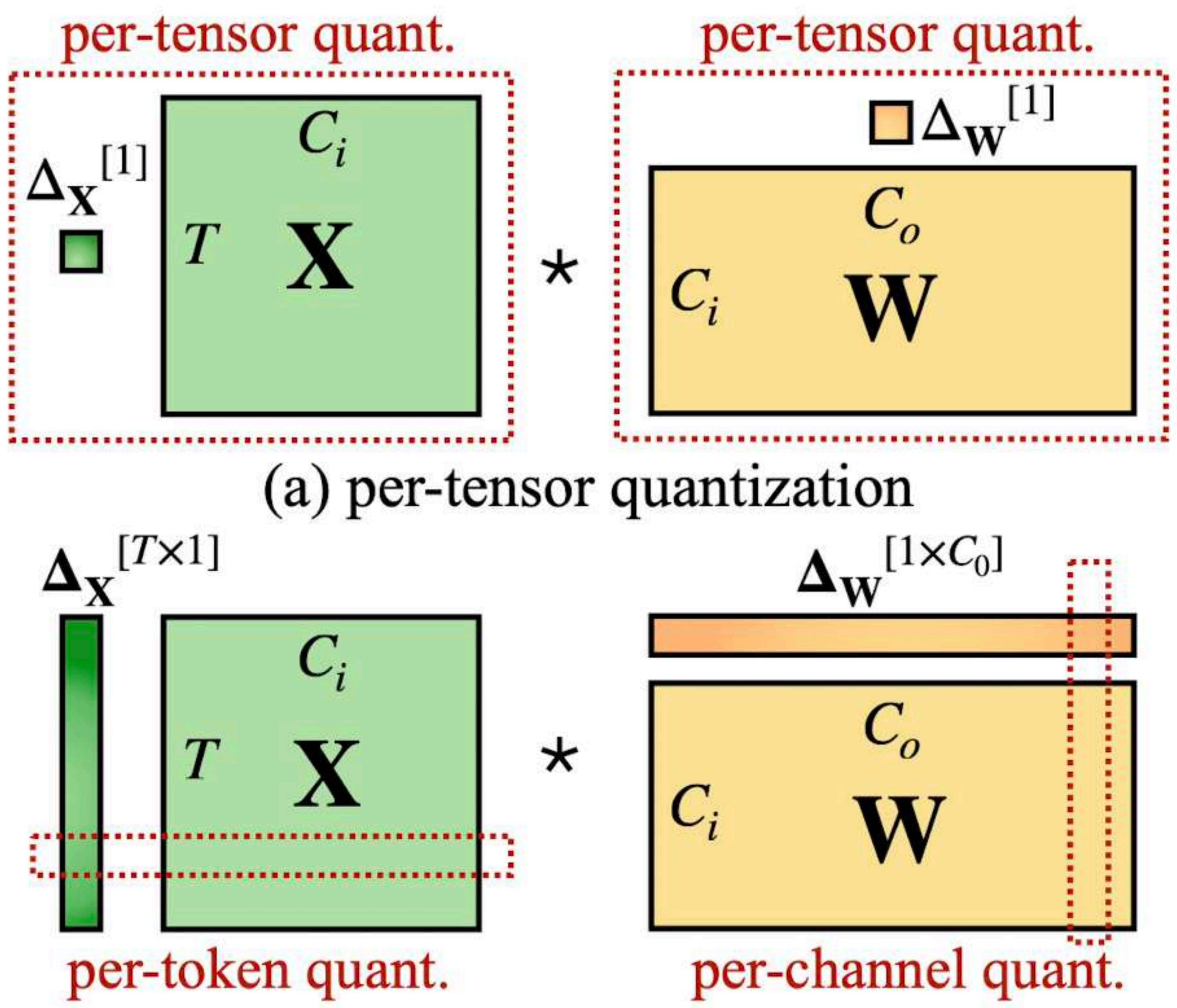
Quantizing a real-valued tensor $\mathbf{x}$ is performed by first mapping it to an unsigned integer grid:

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

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.



(a) per-tensor quantization

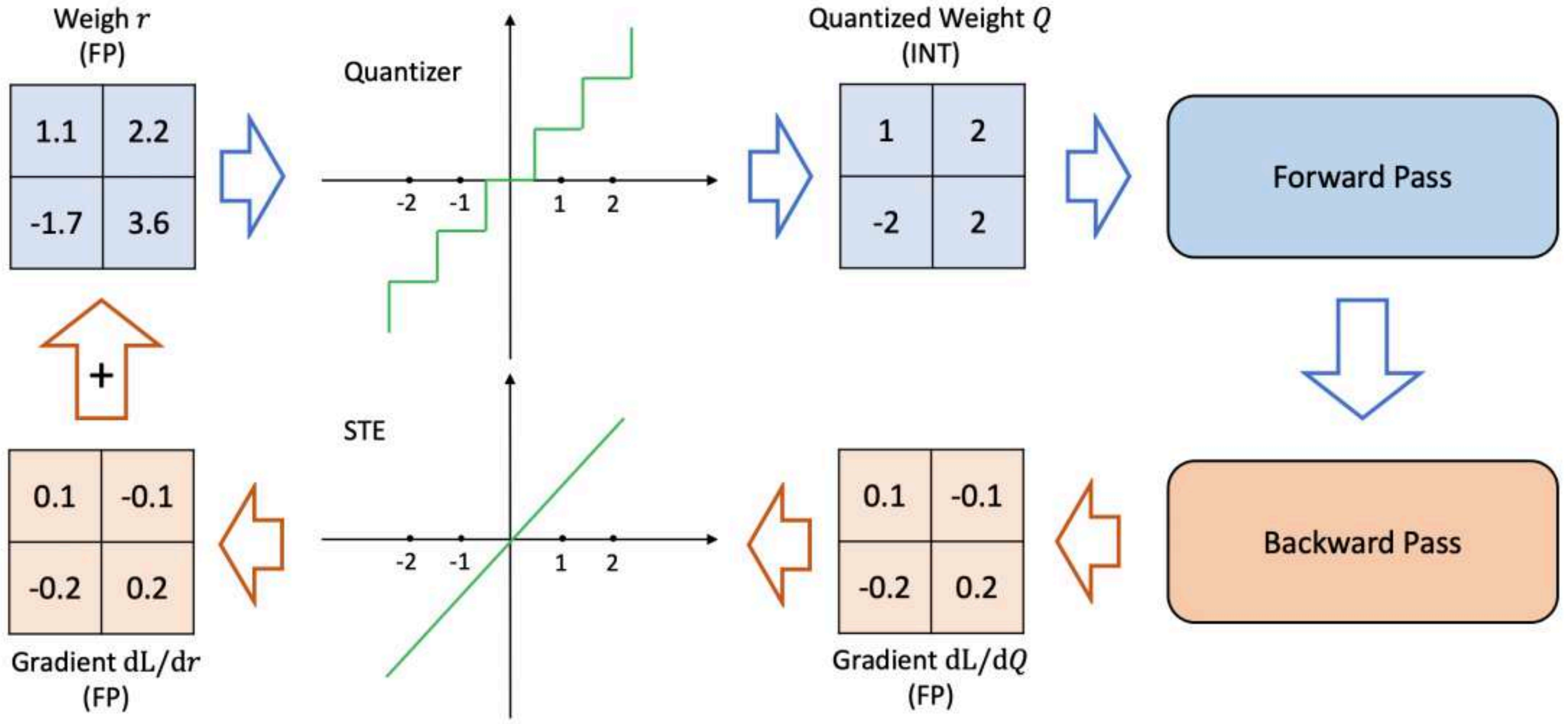
(b) per-token + per-channel quantization

PTQ

- No/~500/1000 calibration samples
- <10 GPU hours
- Calibrating s(scale), z(zero-point)
-

QAT

- Huge dataset
- Better quaity
- Hard to implement



Challenges
Quality drop

- Bert example
- Weights - easy
- Activations - hard
- Outliers - problem

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**

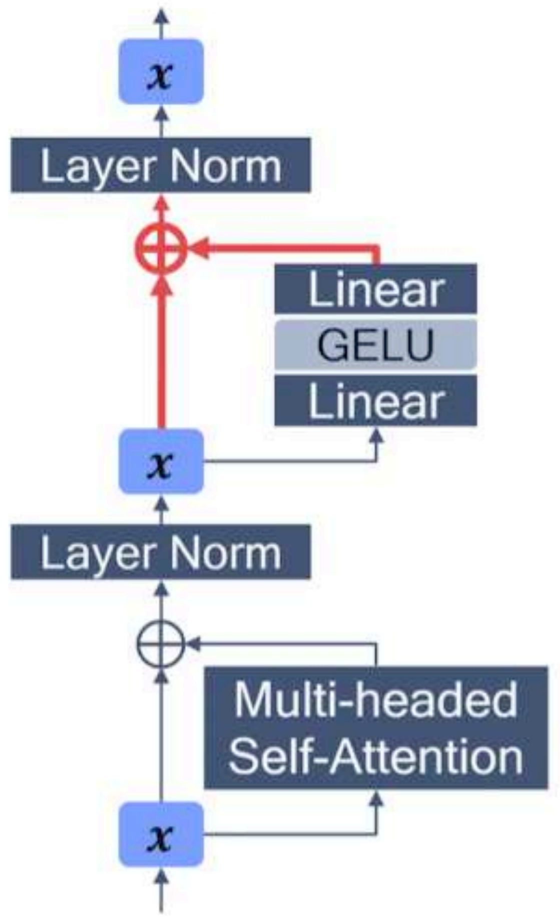
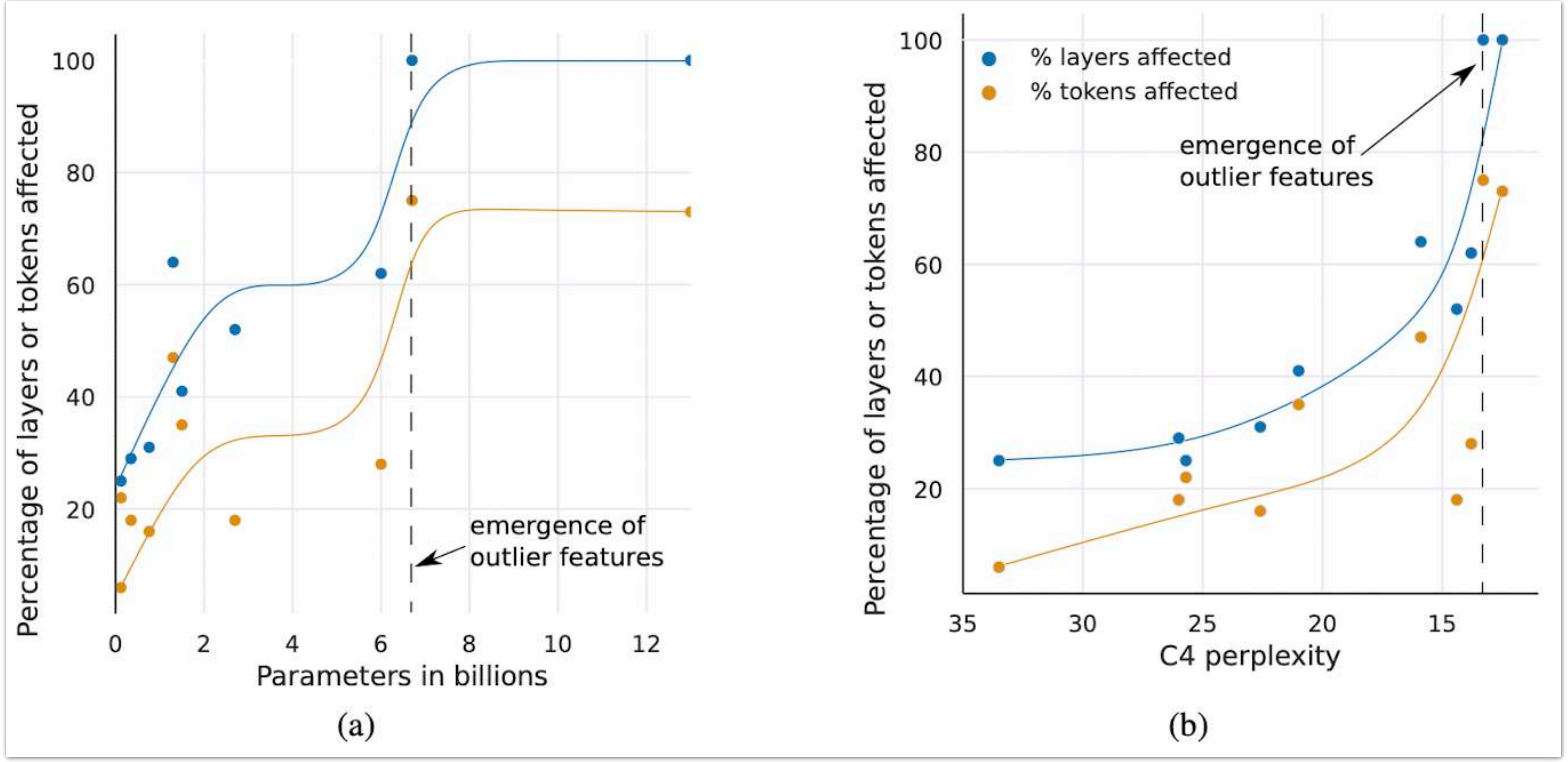


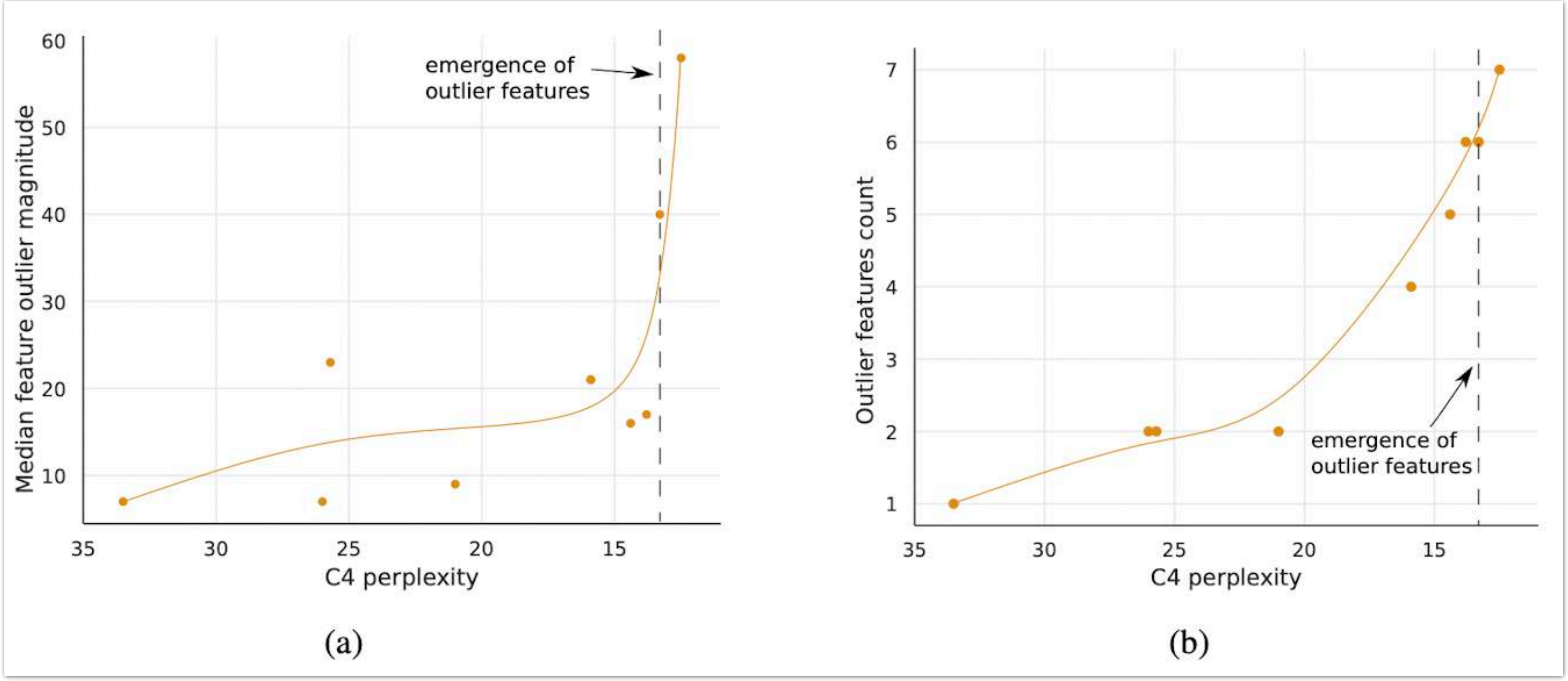
Table 2 & Figure 1: *Left*: Leave-one-out analysis for activation quantizers on problematic GLUE tasks. We set all weights to FP32 and use current min-max (with a batch size of 1) range estimator for activations. We report median score over 5 runs with different random seeds. *Right*: A schematic illustration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

LLM.int8

Outliers is problem

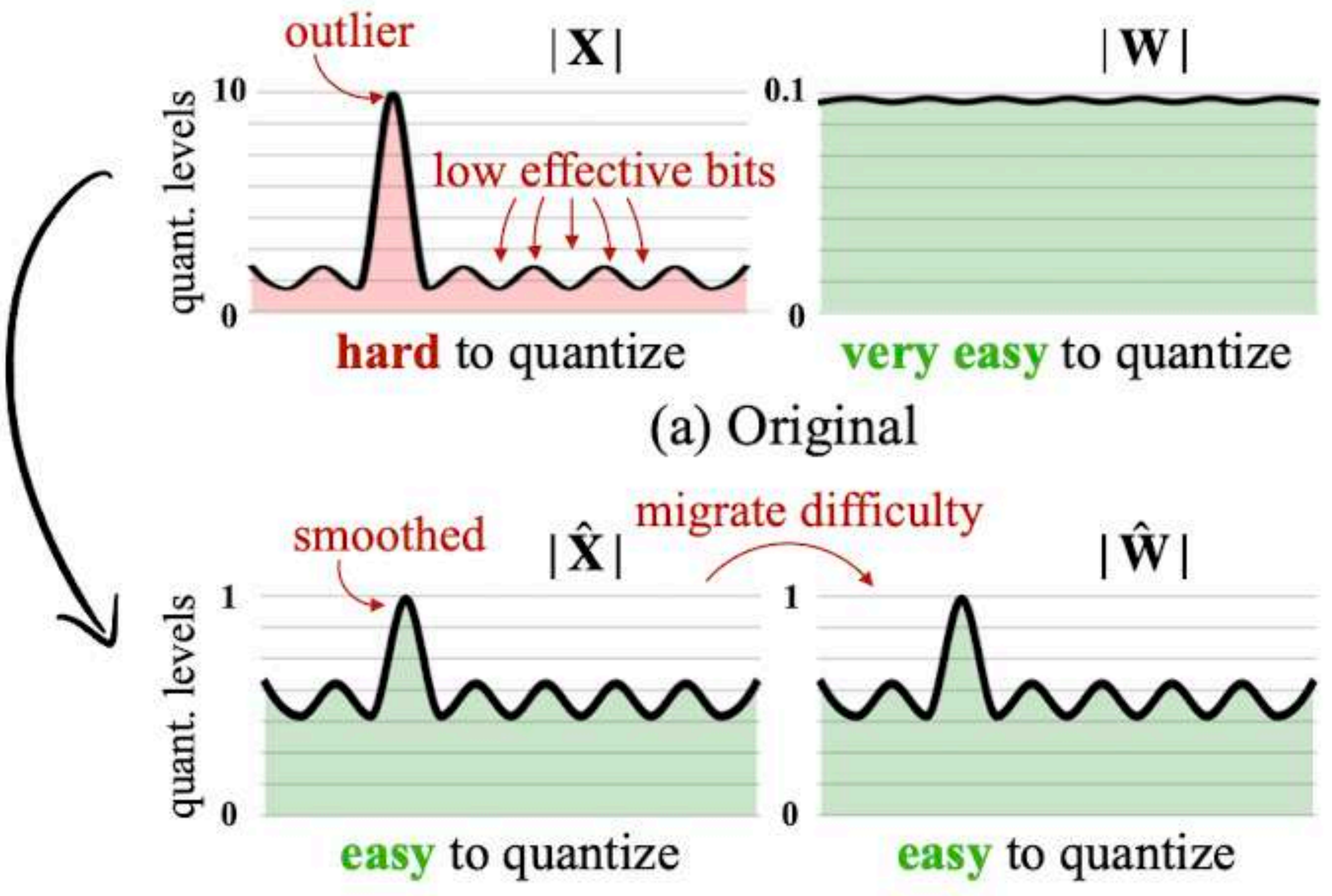
- Outliers occurs in QKVO projections
- Outliers have huge impact on quality





SmoothQuant
Intuition

- Migrate some difficulty to weights



(a) Original

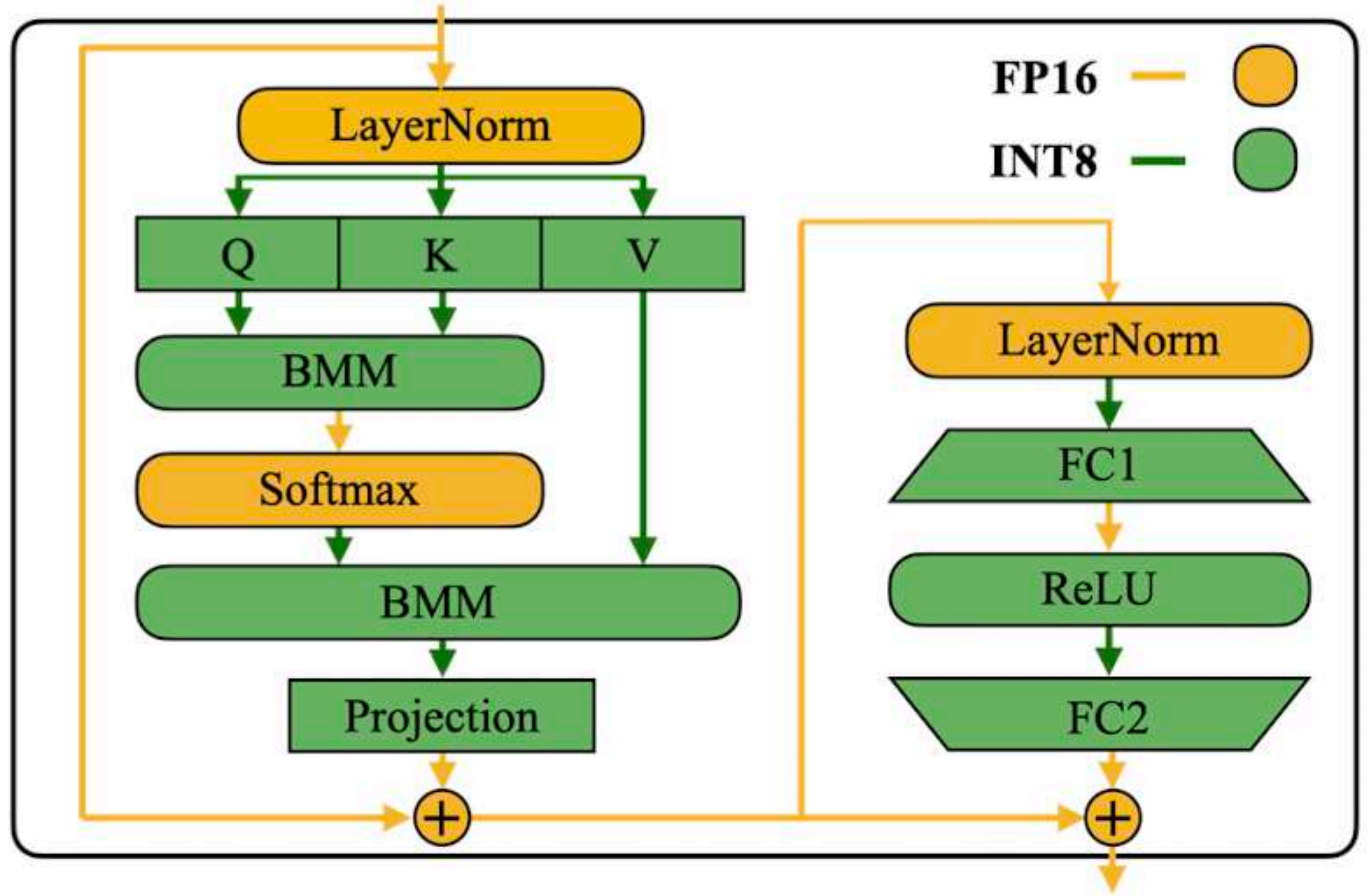
(b) SmoothQuant

SmoothQuant
Method

- W8A8 with same quality
- Apply to all BMM
- Fuse scaling with previous operations

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%



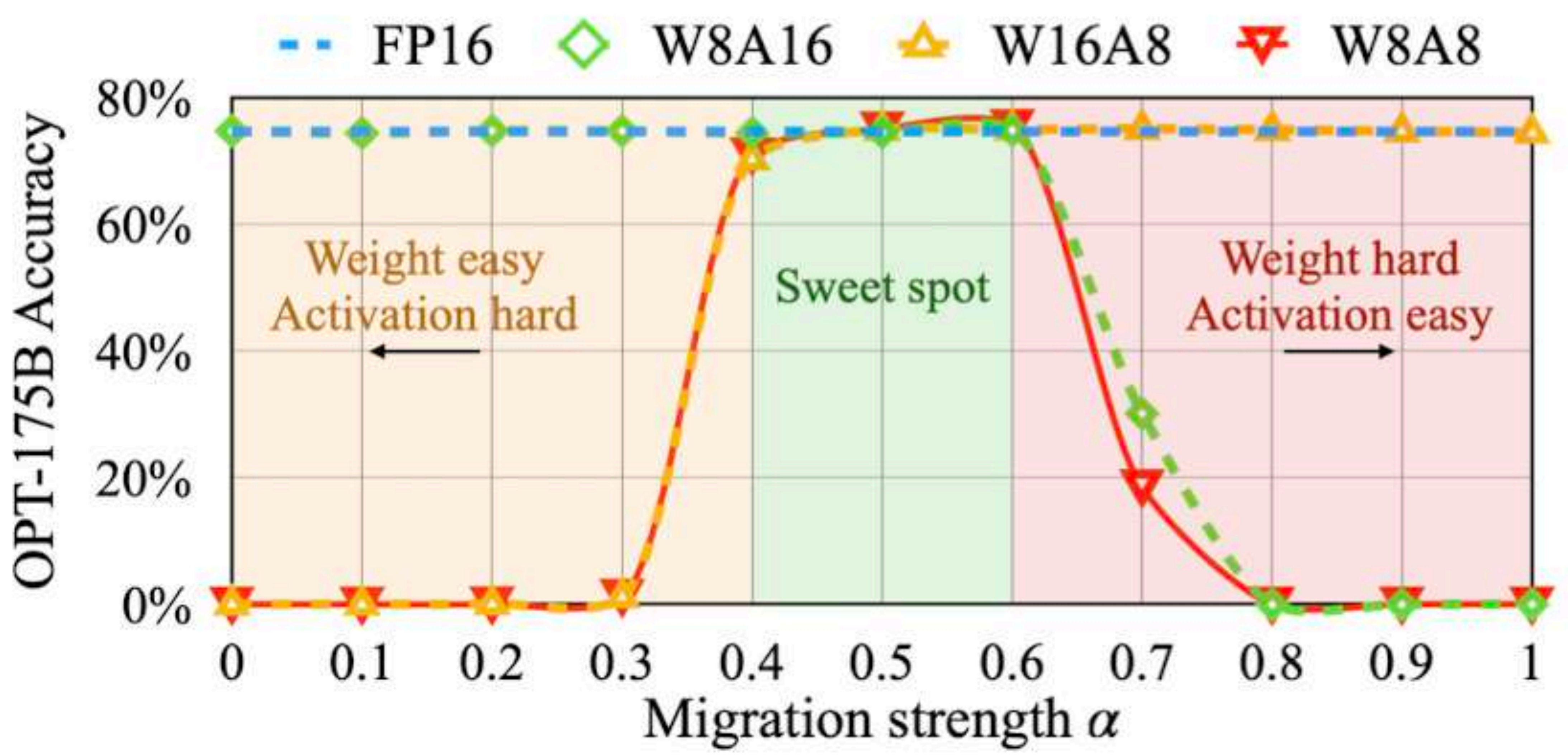
SmoothQuant

Method

$$X^{INT8} = [\frac{X^{FP16}}{s}], s = \frac{max(|X|)}{127}$$

$$Y = (Xdiag(s)^{-1}) \cdot (diag(s)W)$$

$$s_j = \frac{max(|X_J|)^a}{max(|W_j|)^{1-a}}$$

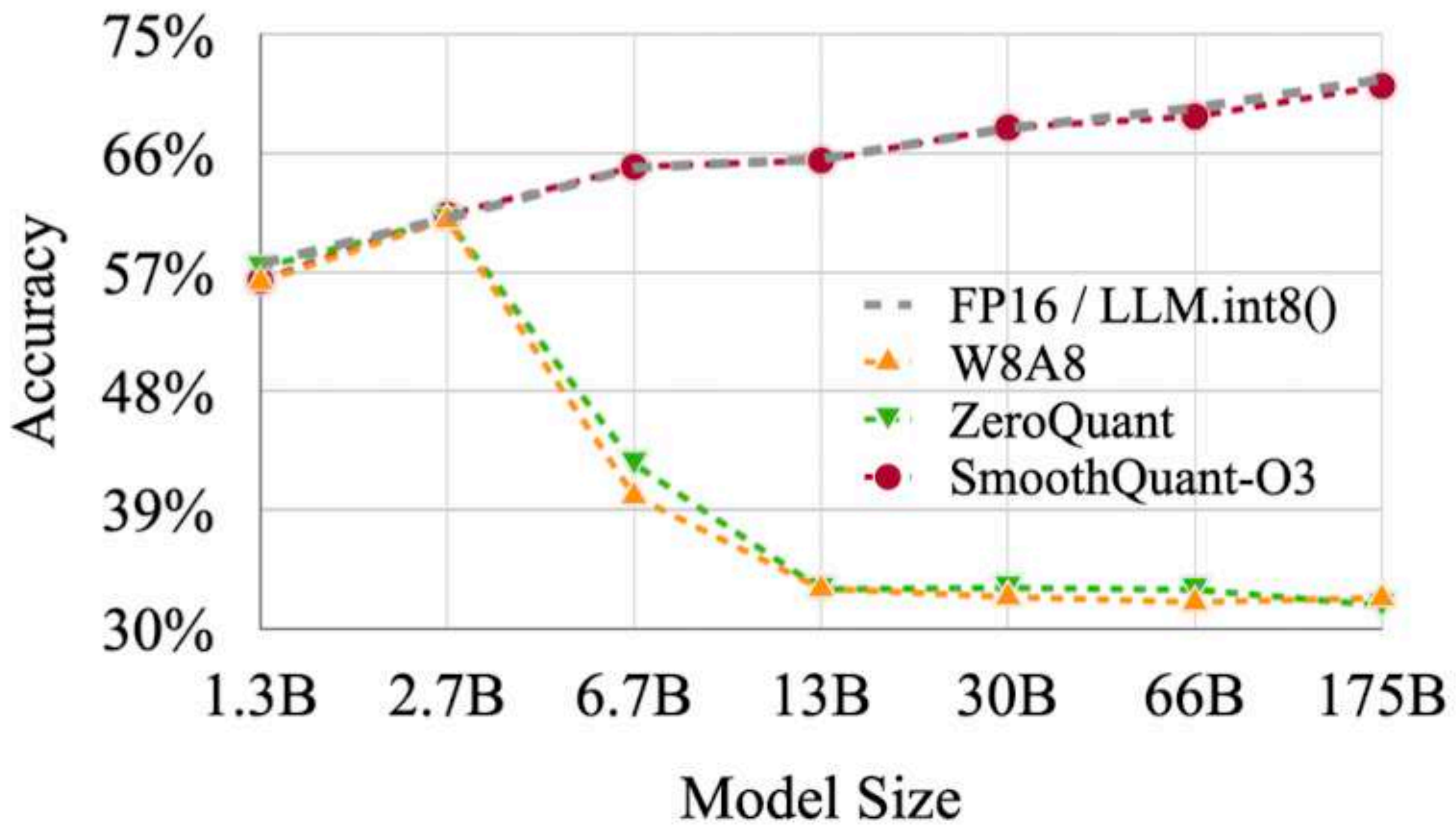


SmoothQuant

Speed

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



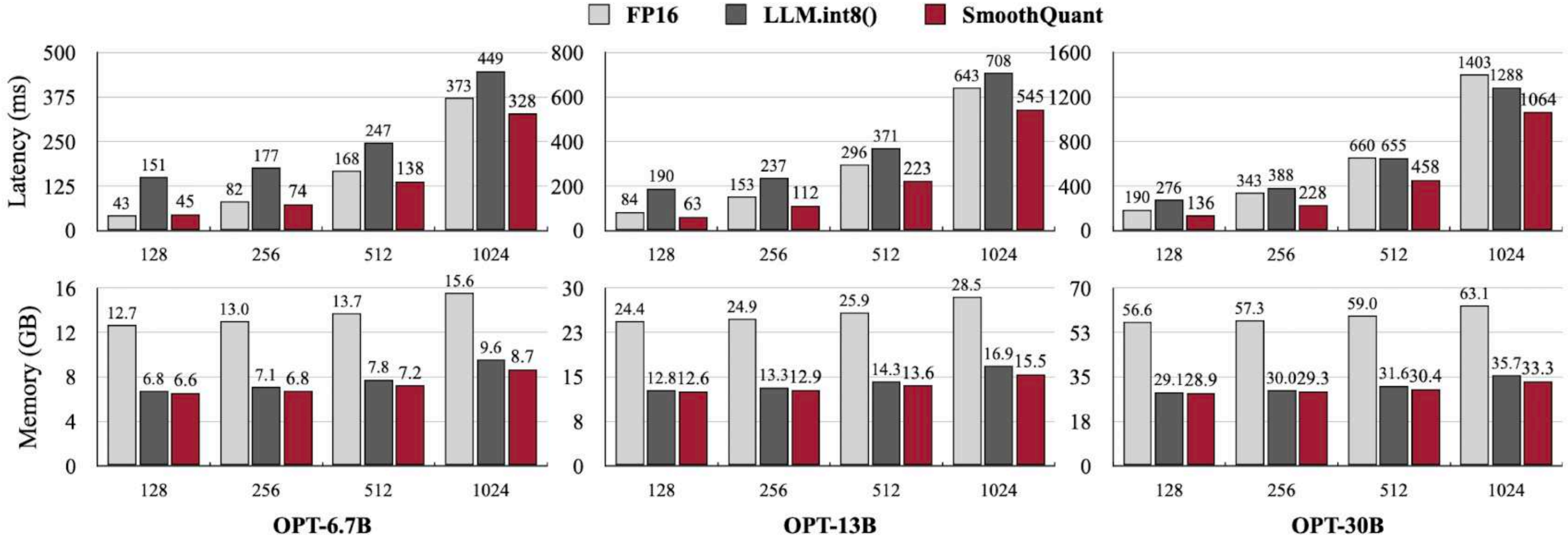


Figure 7: The PyTorch implementation of SmoothQuant-O3 achieves up to **1.51**× speedup and **1.96**× memory saving for OPT models on a single NVIDIA A100-80GB GPU, while `LLM.int8()` slows down the inference in most cases.

GPTQ

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

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

- W4A16
- Decoding only
- Published code and CUDA kernels
- Solving minimization task layer per layer

$$argmin_{\hat{W}} ||WX - \hat{W}X||_2^2$$

Method	Bits	OPT-175B Wiki2	OPT-175B PTB	OPT-175B C4	OPT-175B LAMB. ↑	BLOOM-176B Wiki2	BLOOM-176B PTB	BLOOM-176B C4	BLOOM-176B 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.

Finally

- Есть много методов квантизирующих в 4/8 битов обученные сети почти без потерь
- WxAy самый универсальный сценарий
- Нельзя забывать про diversity калибровочного сэта

Architecture

General

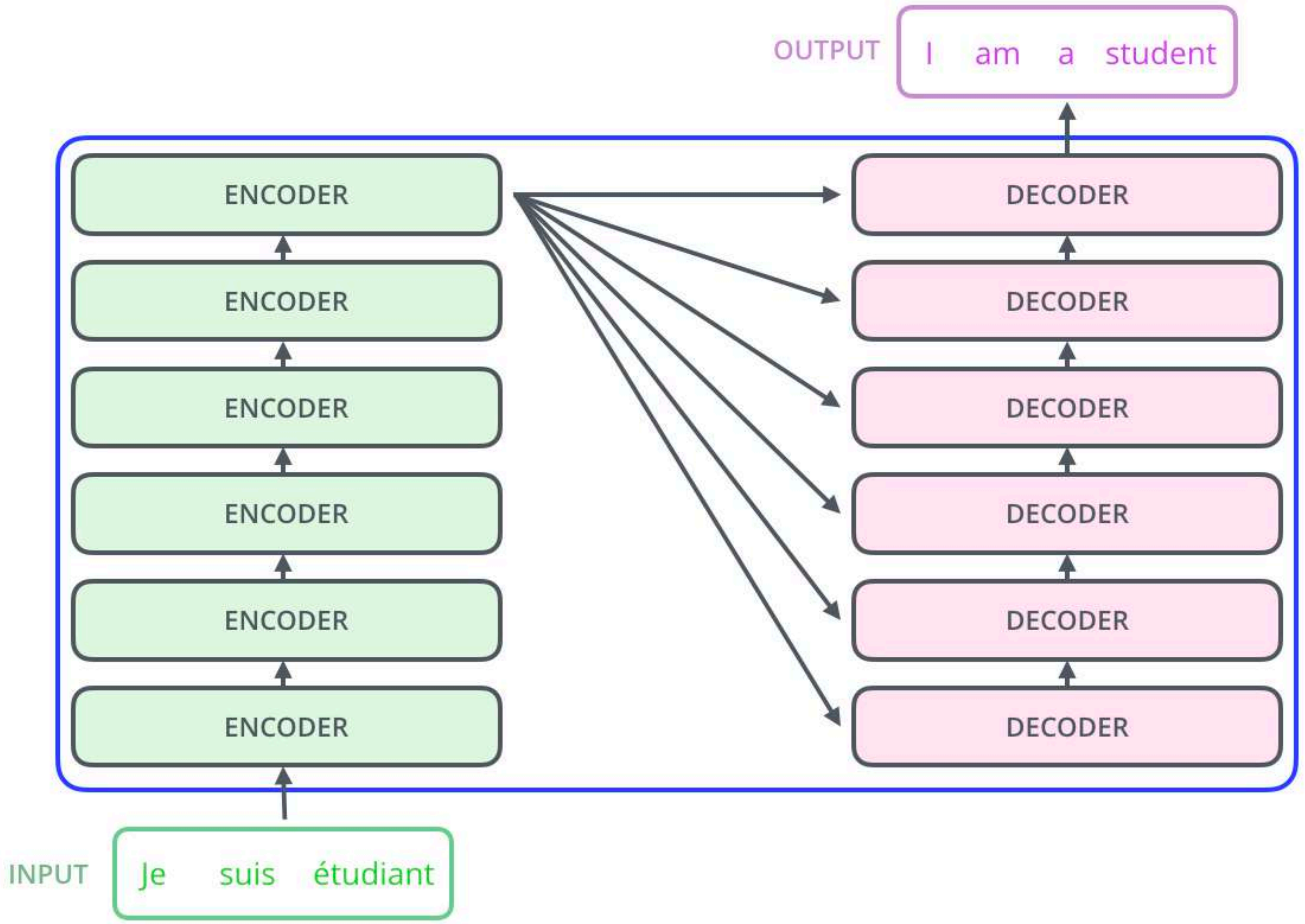
- Как зависит скорость модели от *hidden_size*?

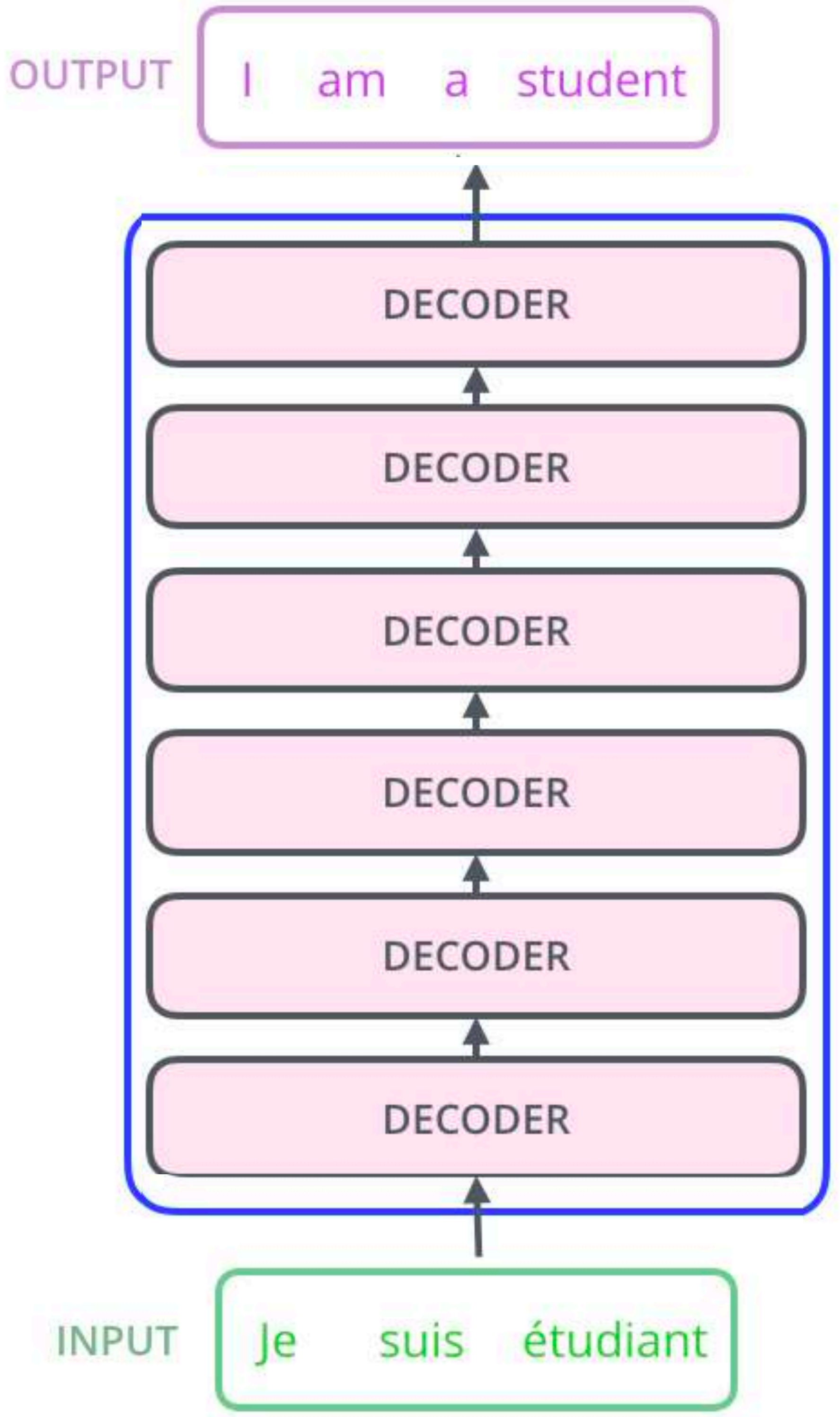
```
qkv: 0.000022 * h^2 + 0.001819 * h + 0.502097
out: 0.000013 * h^2 + 0.001682 * h + 0.826454
ff1: 0.000024 * h^2 + 0.002841 * h + 0.216716
ff2: 0.000023 * h^2 + 0.001856 * h + 0.464659
```

- При $128 \leq h \leq 512$ скорее ~ *h*
- *Тоже самое с seqlen, при seqlen <= 512 скорее ~seqlen*

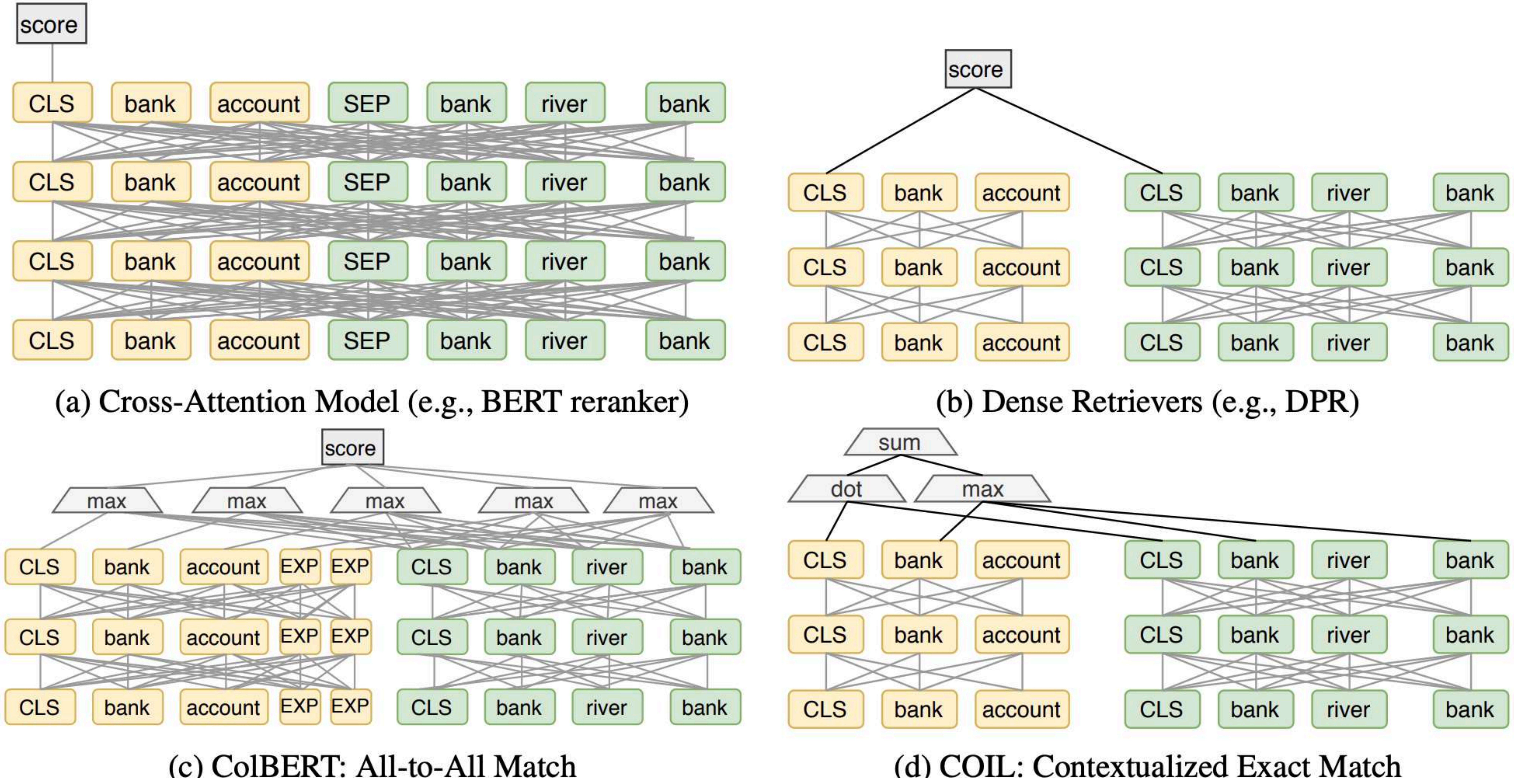
Encoder-decoder vs decoder

Who faster with same amount of parameters ?





Bert tips



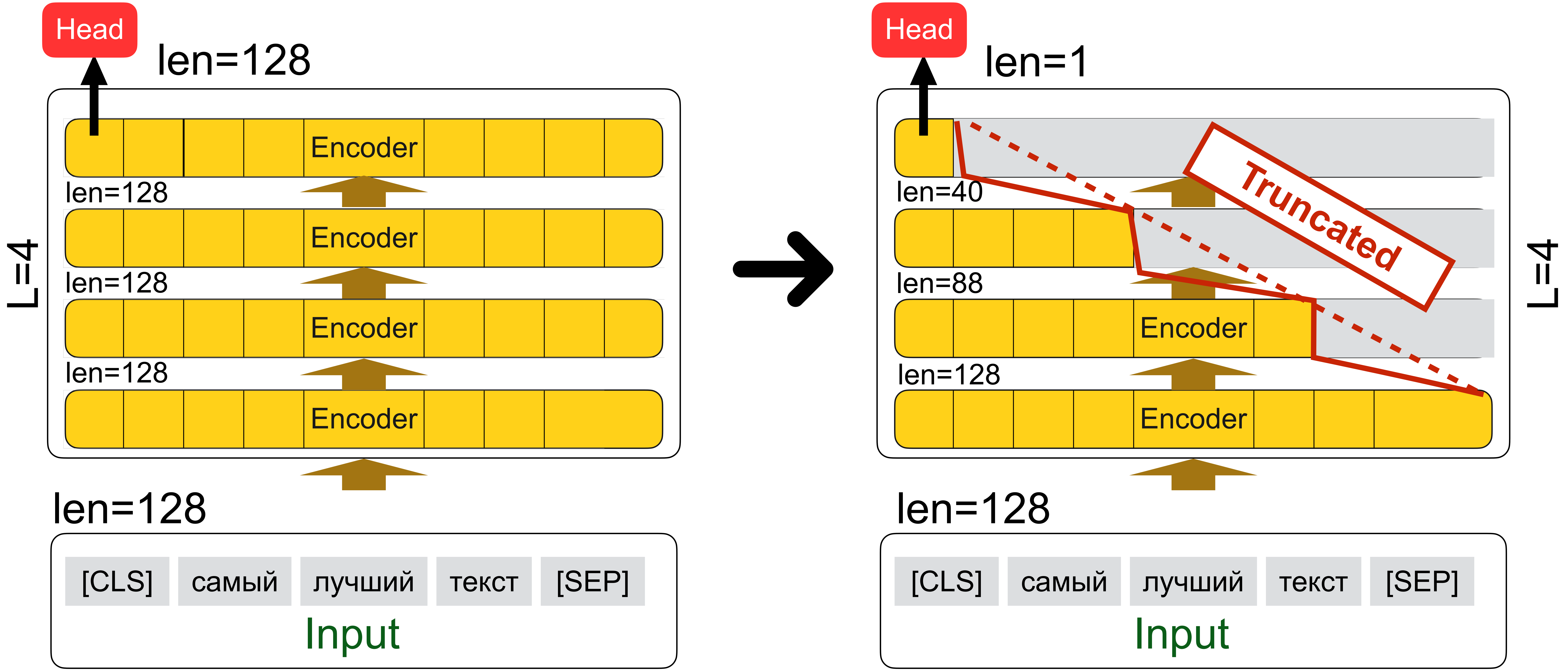
(a) Cross-Attention Model (e.g., BERT reranker)

(b) Dense Retrievers (e.g., DPR)

(c) ColBERT: All-to-All Match

(d) COIL: Contextualized Exact Match

Bert tips


Head
len=128
Encoder
len=128
Encoder
len=128
Encoder
len=128
Encoder
L=4
len=128
[CLS]
самый
лучший
текст
[SEP]
Input
Head
len=1
len=40
len=88
Encoder
len=128
Encoder
Truncated
L=4
len=128
[CLS]
самый
лучший
текст
[SEP]
Input

Speculative decoding

Speculative decoding

Vanilla idea

- Small draft model, e.g 7B
- Large verification model, e.g. 70B
- Decode 1 to K+1 tokens per iteration
- Verify by large model

Table 1 | **Chinchilla performance and speed on XSum and HumanEval with naive and speculative sampling at batch size 1 and $K = 4$.** XSum was executed with nucleus parameter $p = 0.8$, and HumanEval with $p = 0.95$ and temperature 0.8.

Sampling Method	Benchmark	Result	Mean Token Time	Speed Up
ArS (Nucleus)	XSum (ROUGE-2)	0.112	14.1ms/Token	1×
SpS (Nucleus)		0.114	7.52ms/Token	1.92×
ArS (Greedy)	XSum (ROUGE-2)	0.157	14.1ms/Token	1×
SpS (Greedy)		0.156	7.00ms/Token	2.01×
ArS (Nucleus)	HumanEval (100 Shot)	45.1%	14.1ms/Token	1×
SpS (Nucleus)		47.0%	5.73ms/Token	2.46×

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

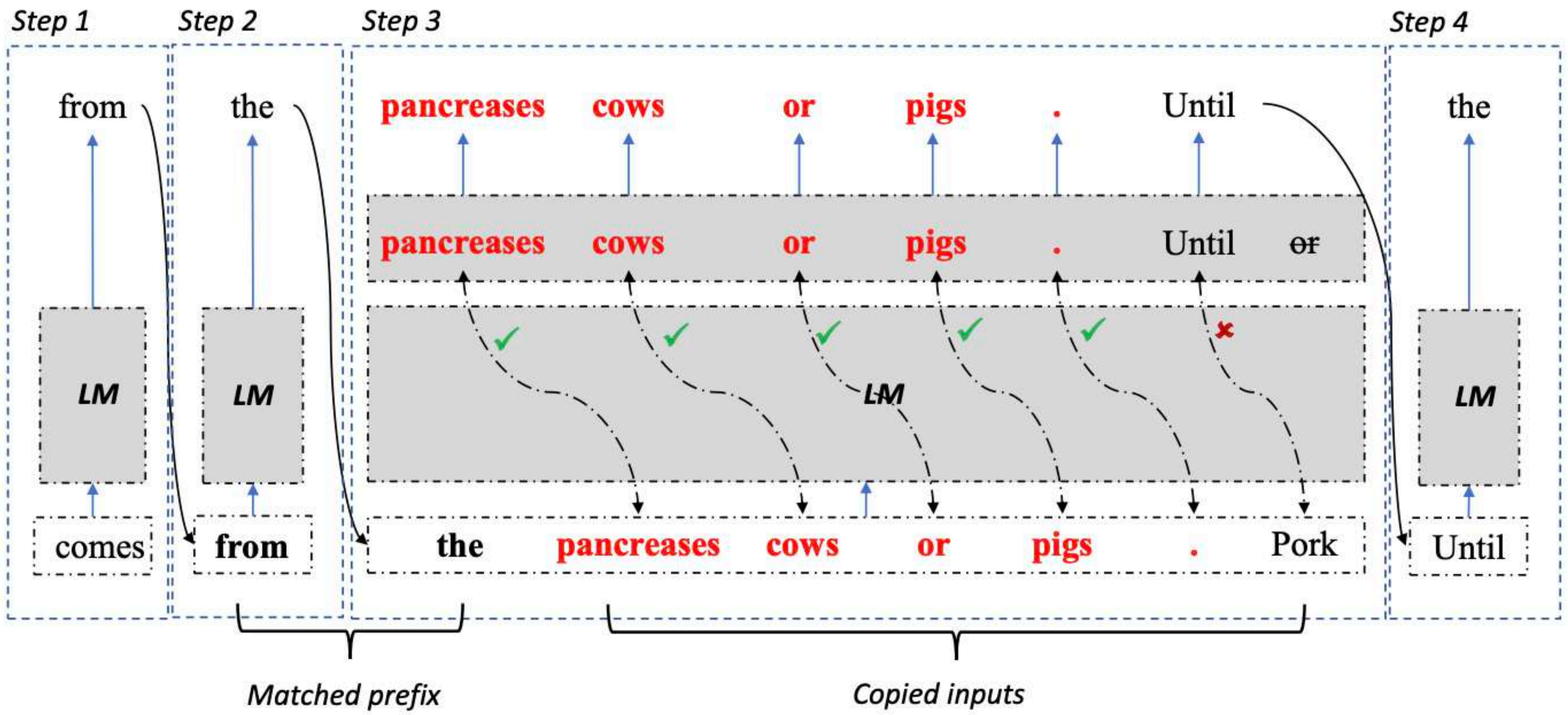
LLMA

Inference with reference

Algorithm 1 LLMA Decoding.

Input: $\boldsymbol{x}$, $\boldsymbol{D} = (\boldsymbol{d}_1, \ldots, \boldsymbol{d}_n)$, n, k, N;
Output: $\boldsymbol{y}$;

$\boldsymbol{y} \leftarrow []$
while $\text{LEN}(\boldsymbol{y}) < N$ **do**
  $matched, \boldsymbol{d}, pos \leftarrow \text{MATCH\_NGRAMS}(\boldsymbol{y}, \boldsymbol{D}, n)$
  if $\neg matched$ **then**
    $o = \text{LLM}(\boldsymbol{x}, \boldsymbol{y})$
    $\text{APPEND}(\boldsymbol{y}, o)$
    continue
  end if
  $(o_0, o_1, \ldots, o_k) \leftarrow \text{LLM}(\boldsymbol{x}, \boldsymbol{y}, d_{pos}, \ldots, d_{pos+k-1})$
  $\text{APPEND}(\boldsymbol{y}, o_0)$
  for i **in** $0, \ldots, k-1$ **do**
    if o_i **neq** d_{pos+i} **then**
      break
    end if
    $\text{APPEND}(\boldsymbol{y}, o_{i+1})$
  end for
end while



Why speculative sampling is bad

- Optimize only MBU
- Work only with batch_size=1
- Don’t optimize memory(otherwise)
- hard to implement

Conclusion

Conclusion

- Don’t forget about MBU vs MFU!!!
- Smoothquant is universal, gptq/aqlm if you have small rps/need to host huge models on 1 GPU
- If you have reword model use it!, otherwise hard-label KD if you have a lot of compute, slim if not
- Don’t forget about architecture optimizing