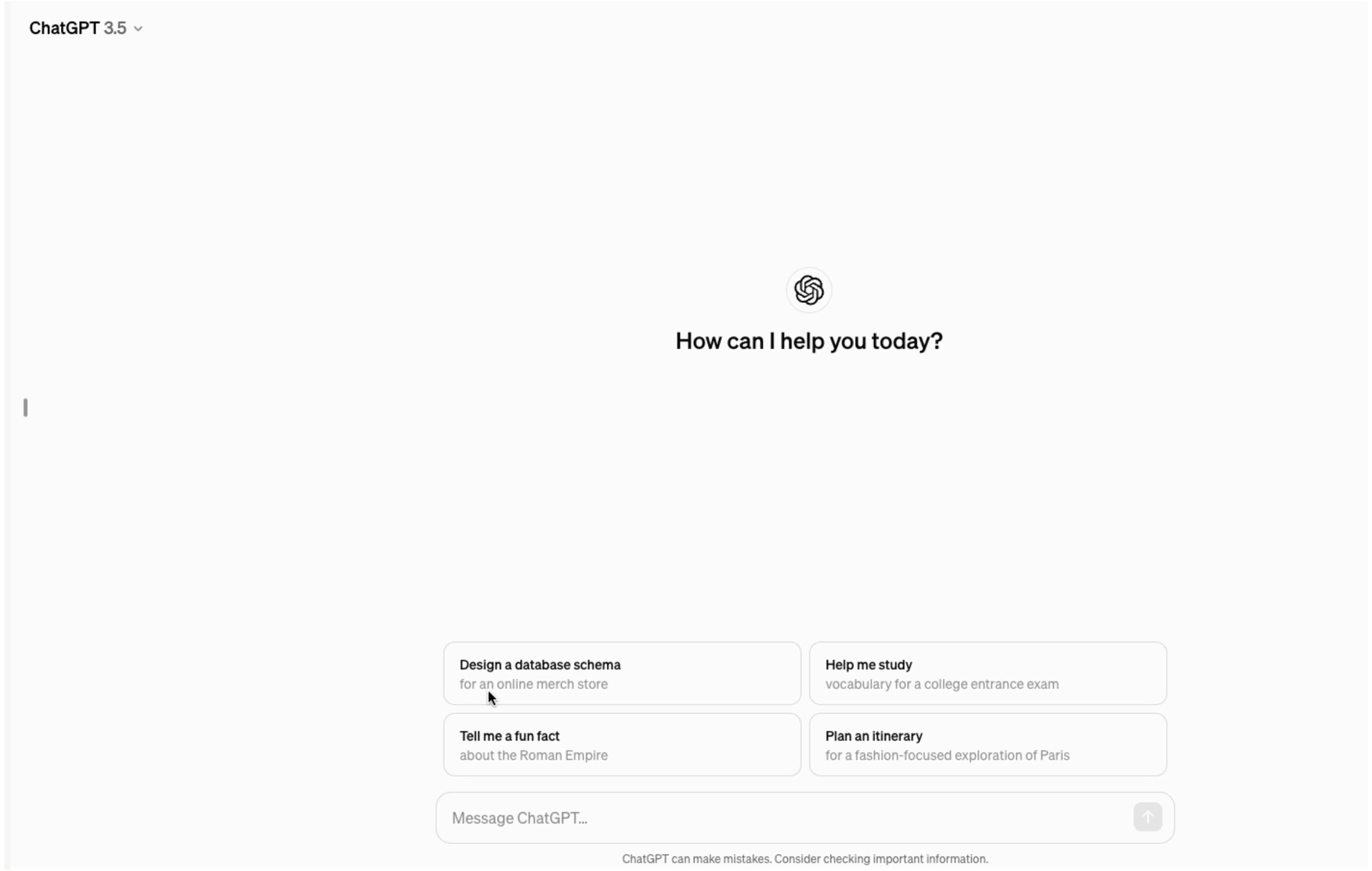
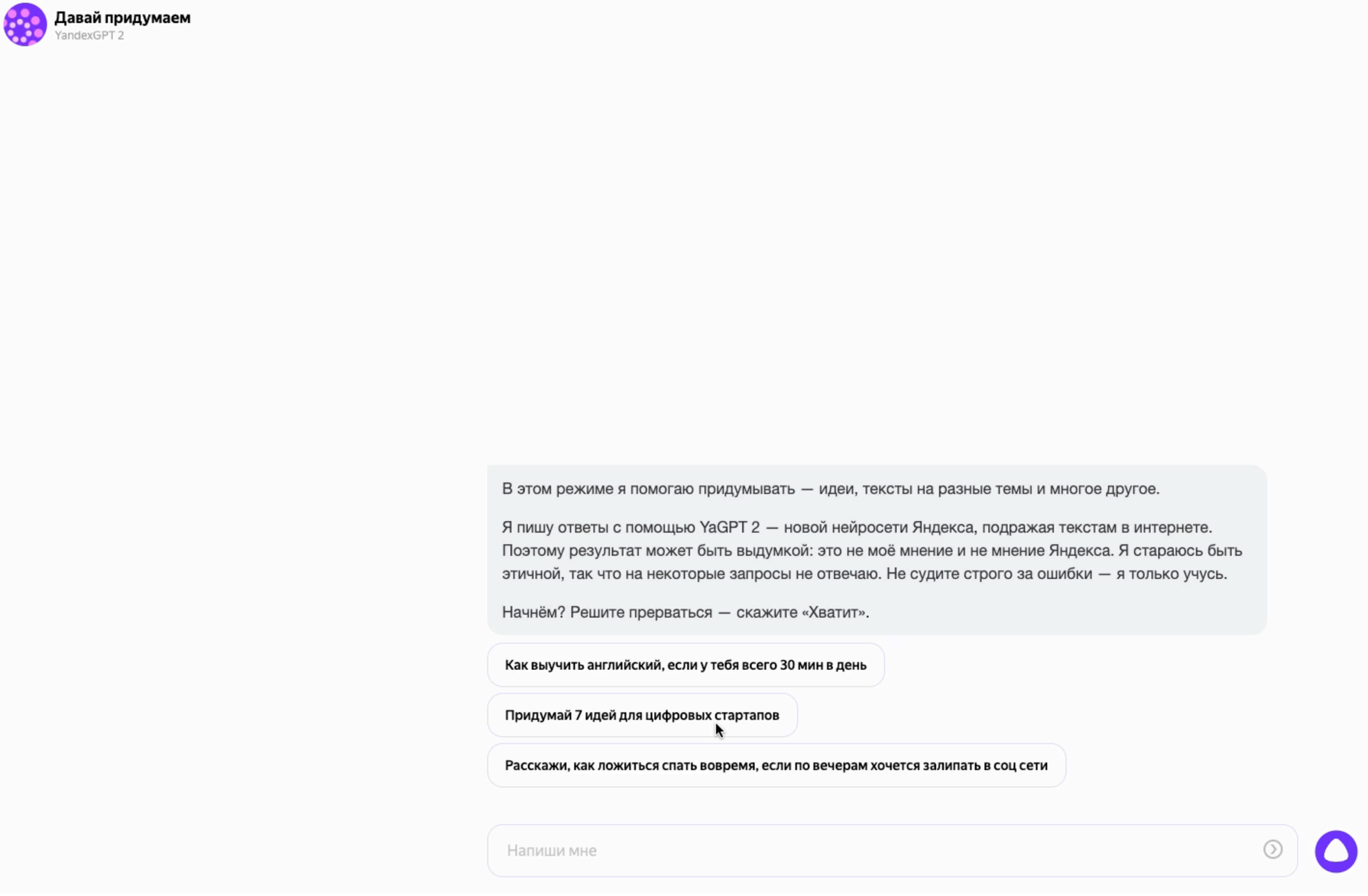
LLM Inference

Efficient DL, Episode VIII ’25

ChatGPT


ChatGPT 3.5
How can I help you today?
Design a database schema
for an online merch store
Help me study
vocabulary for a college entrance exam
Tell me a fun fact
about the Roman Empire
Plan an itinerary
for a fashion-focused exploration of Paris
Message ChatGPT...
ChatGPT can make mistakes. Consider checking important information.

YaGPT


Давай придумаем
YandexGPT 2
В этом режиме я помогаю придумывать — идеи, тексты на разные темы и многое другое.
Я пишу ответы с помощью YaGPT 2 — новой нейросети Яндекса, подражая текстам в интернете. Поэтому результат может быть выдумкой: это не моё мнение и не мнение Яндекса. Я стараюсь быть этичной, так что на некоторые запросы не отвечаю. Не судите строго за ошибки — я только учусь.
Начнём? Решите прерваться — скажите «Хватит».
Как выучить английский, если у тебя всего 30 мин в день
Придумай 7 идей для цифровых стартапов
Расскажи, как ложиться спать вовремя, если по вечерам хочется залипать в соц сети
Напиши мне

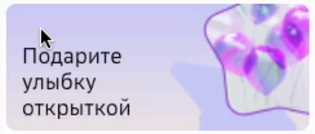
GigaChat

— Салют, вы в GigaChat!
И это не просто очередная текстовая модель 😀
GigaChat очень трудолюбив и может сказать и нарисовать почти всё, что вы скажете. Пользуйтесь с умом 🧠

> Начиная работу с GigaChat, вы берете на себя ответственность за соблюдение законодательства РФ и общепризнанных правил этики в соответствии с правилами использования сервиса. Ознакомьтесь с ними!

— Совет: Используйте кнопку «Новый чат», если хотите сменить тему разговора 💬

— Если не знаете с чего начать, загляните в наш гайд, там есть не только советы, но и примеры для вдохновения.



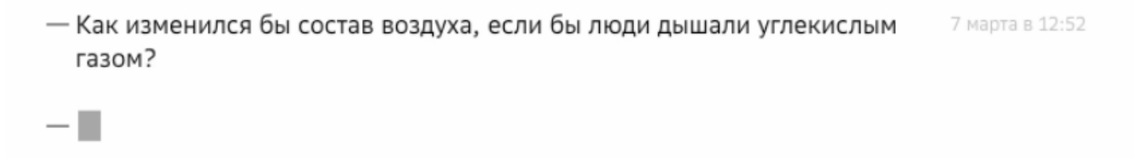
Как изменился бы состав воздуха, если бы люди дышали углекислым газом?

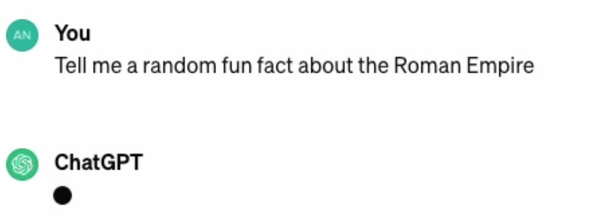
Напиши резюме для начинающего PR-специалиста

Спросите меня о чём-нибудь

TTFT

Time To First Token

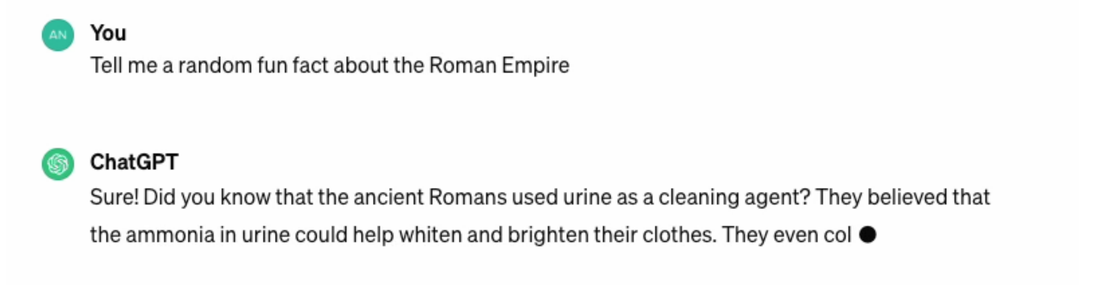


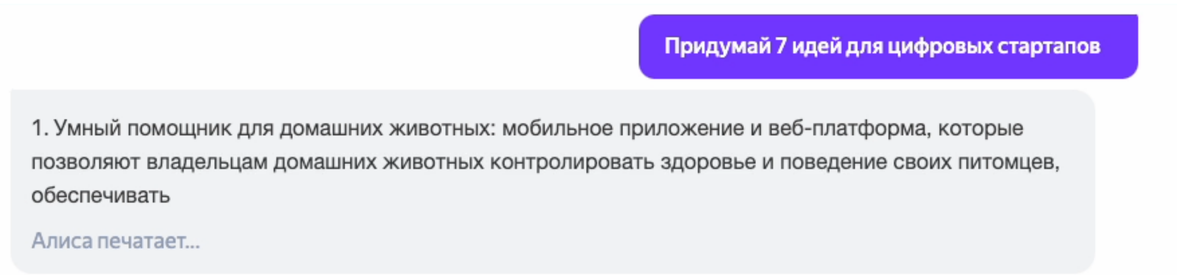


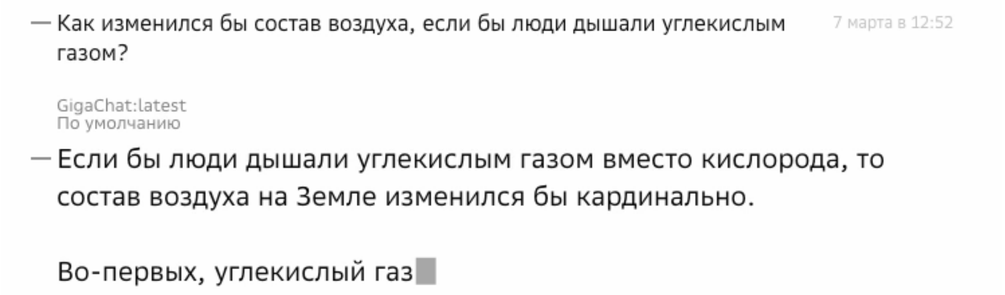
Context decoding

TPS

Tokens Per Second

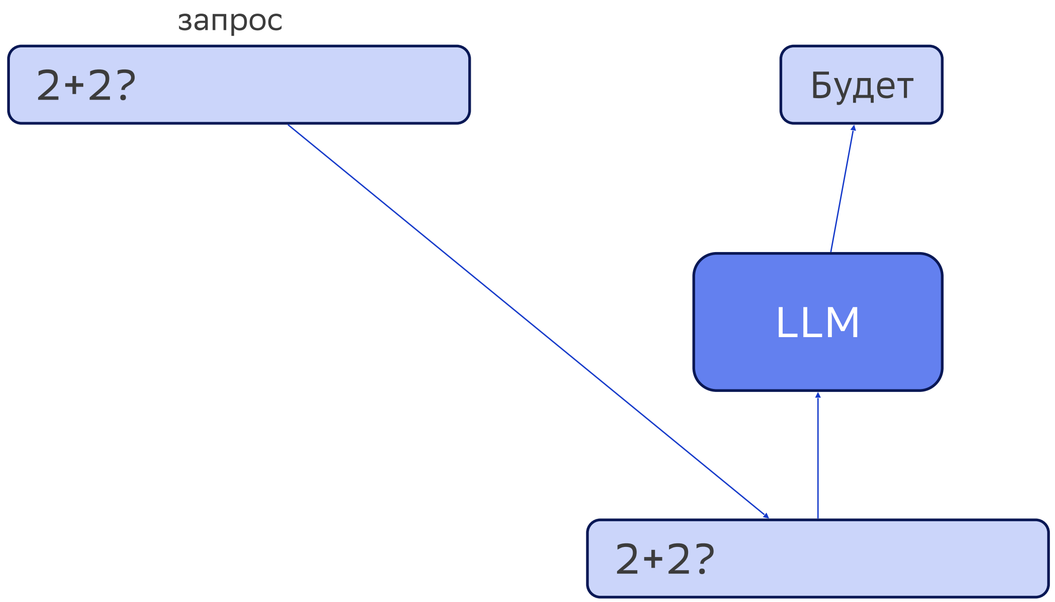




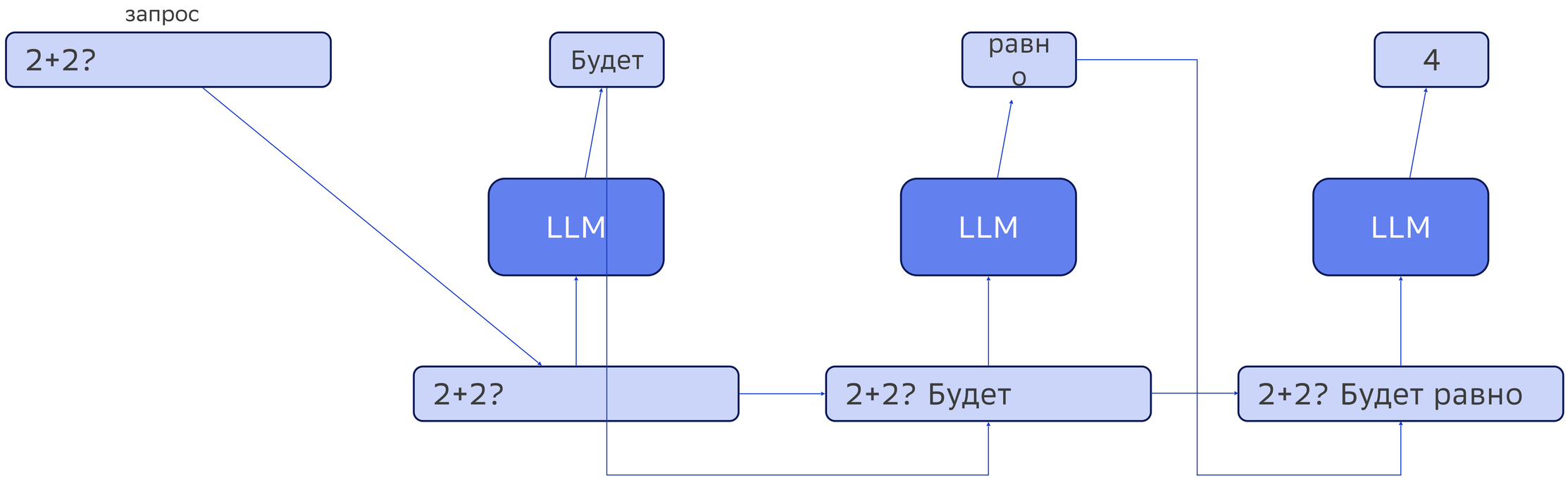


Generation step

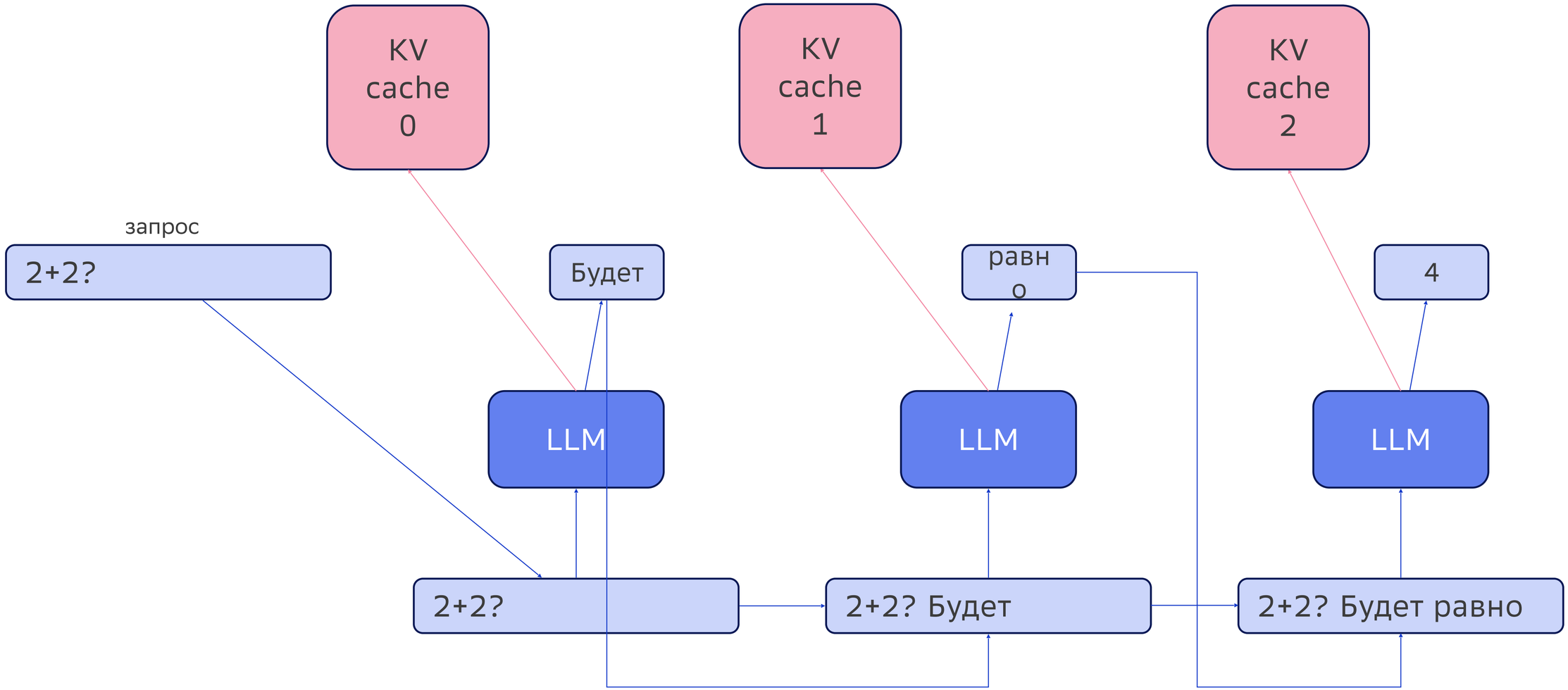
Обработка запроса



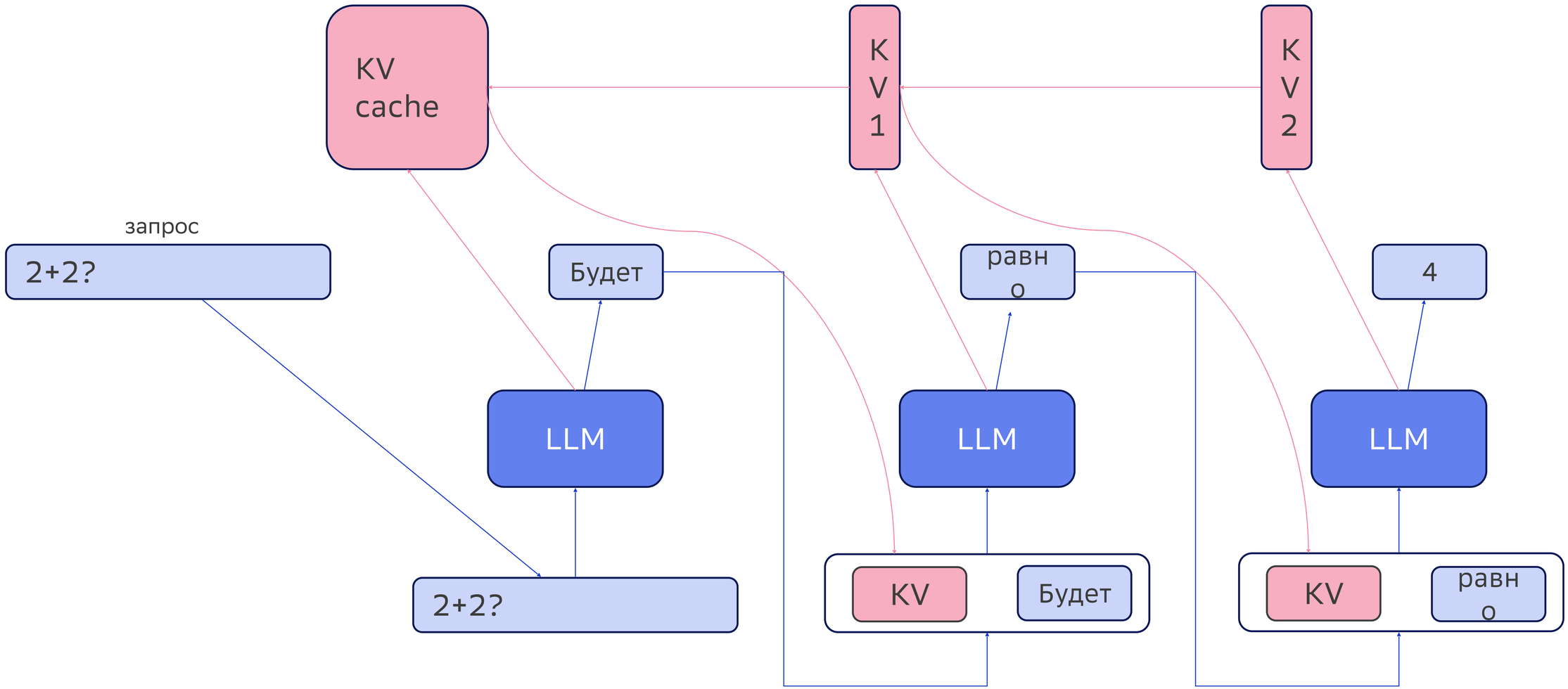
Обработка запроса



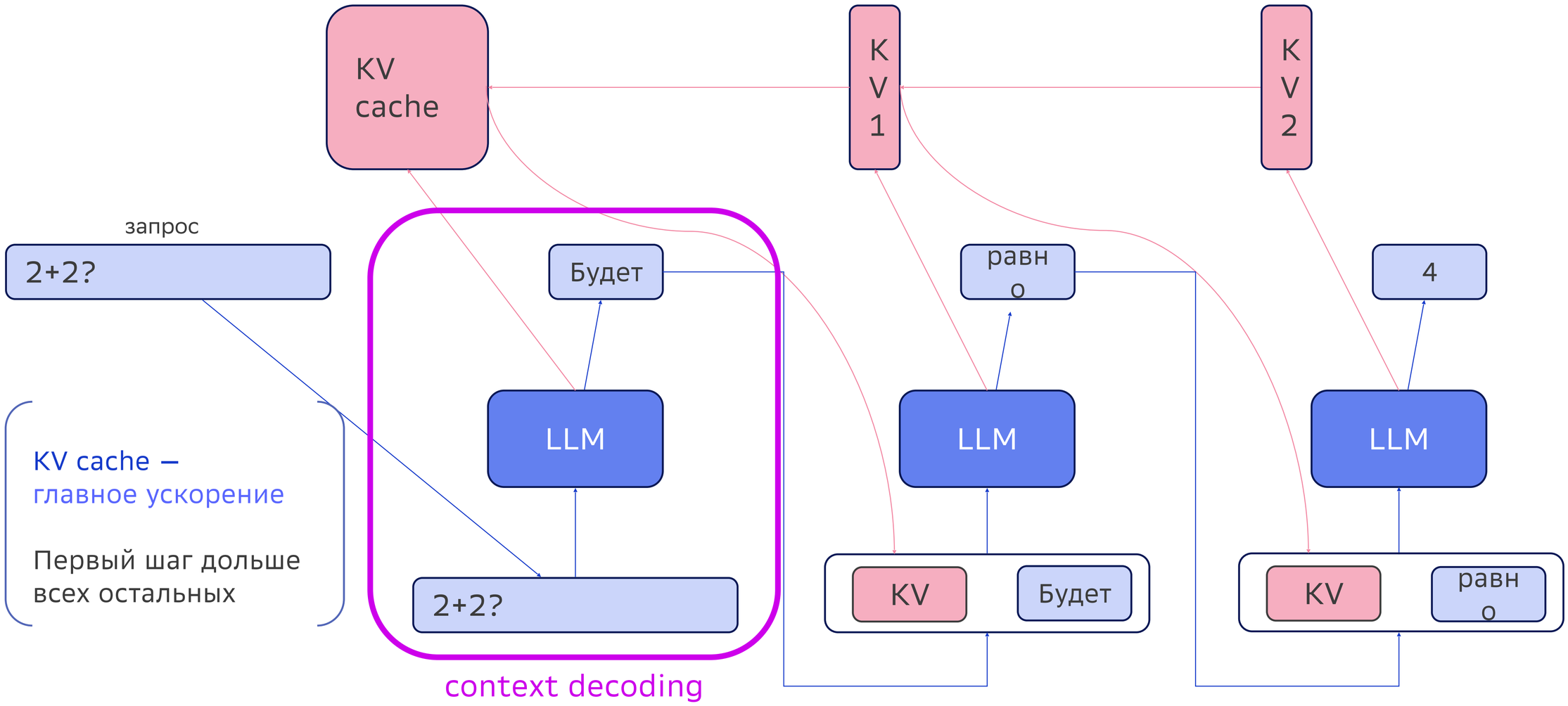
Обработка запроса



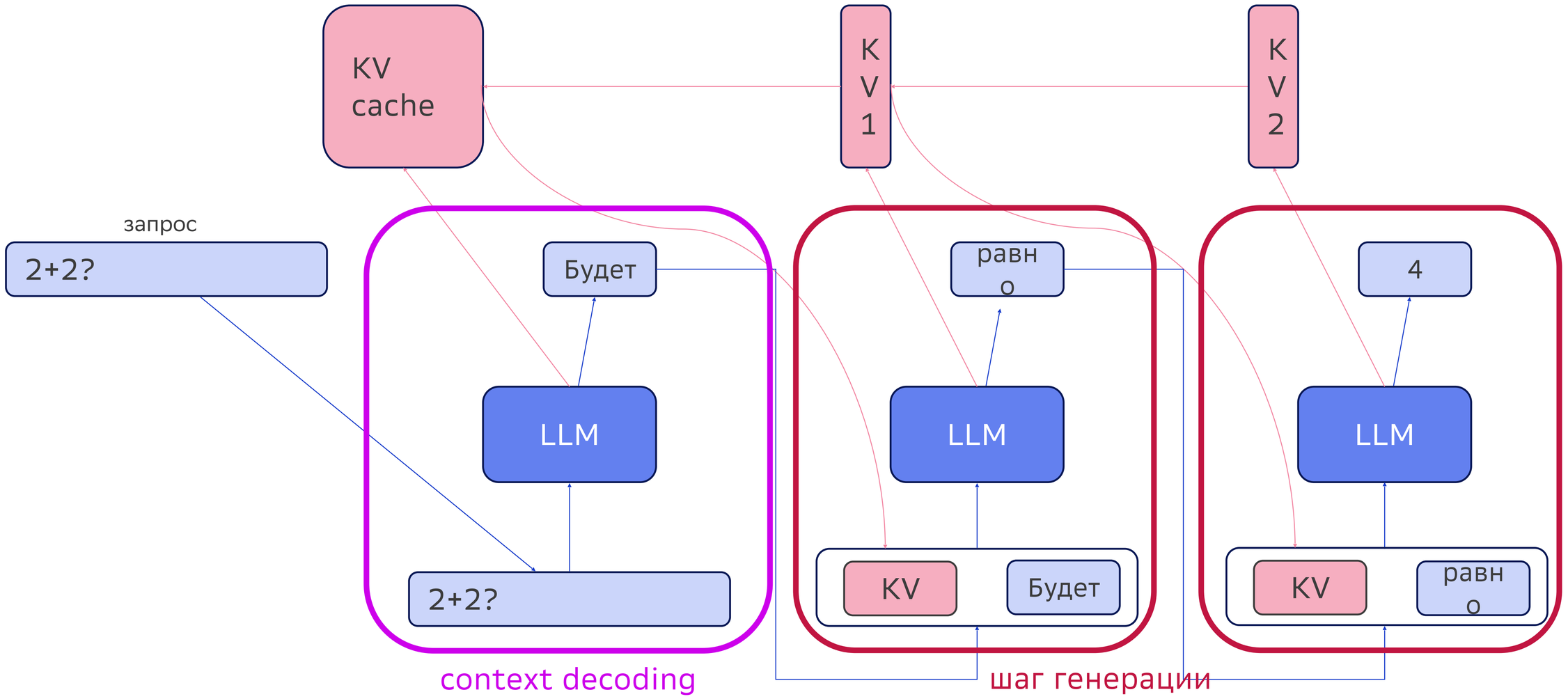
Обработка запроса



Обработка запроса



Обработка запроса

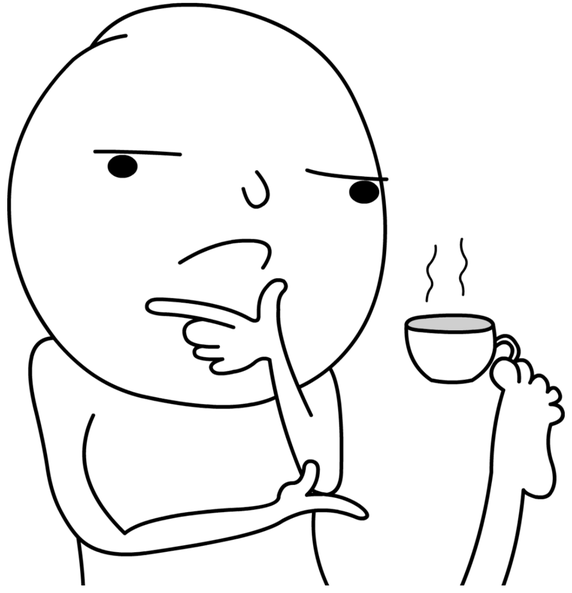


TPS

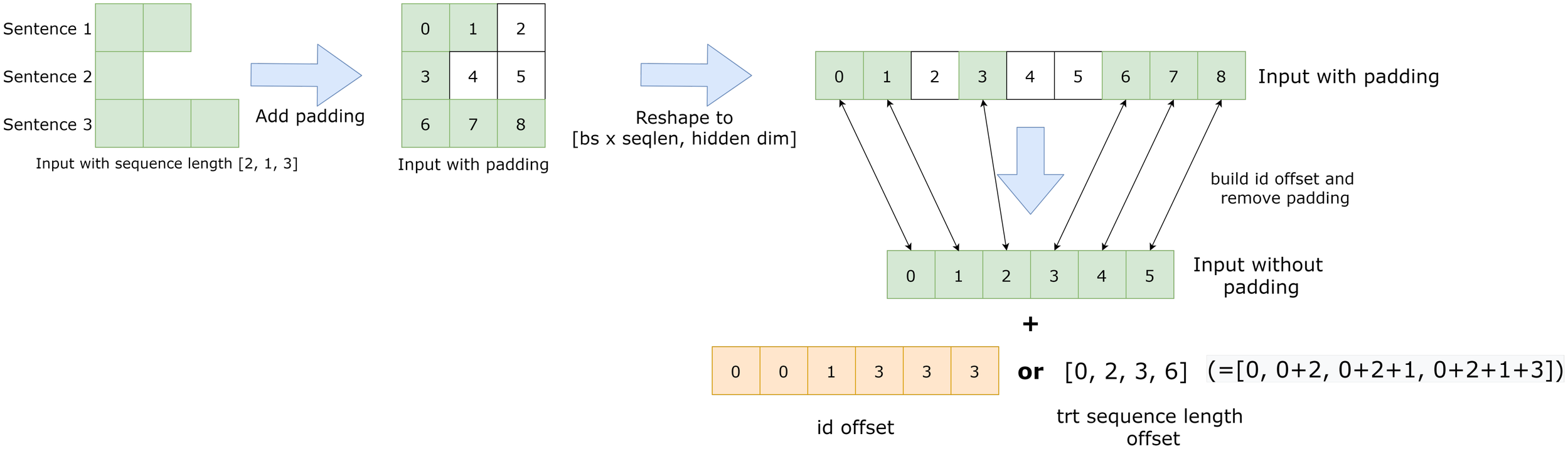
Tokens Per Second

	TPS for single query	TPS for single instance
Assistant	40 TPS is OK (or is it?)	Economics
Offline	Do we care?	Most efficient way

What about batches?

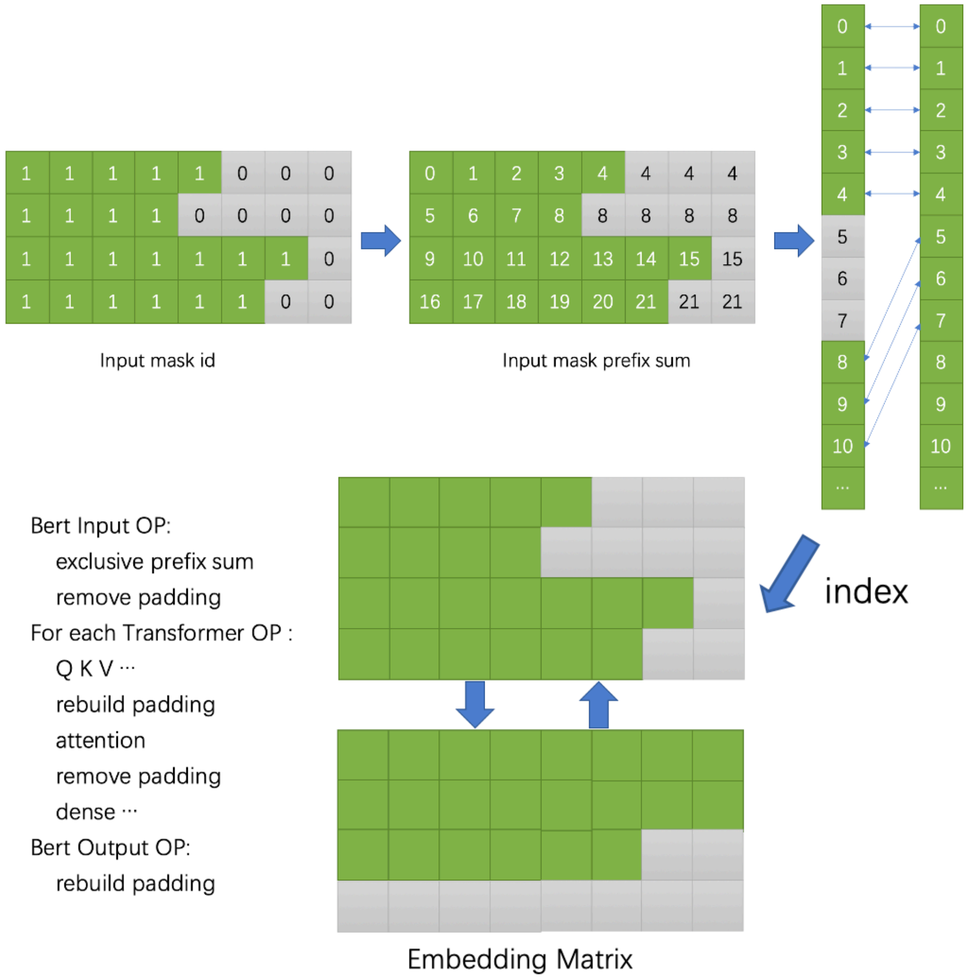


Batch



Source: https://github.com/bytedance/effective_transformer

Batch



Source: https://github.com/bytedance/effective_transformer

Batch

Tesla V100, float16, maximum sequence length=64, average sequence length≈40

batch_size	XLA (in ms)	Faster Transformer (in ms)	Speedup over XLA	Effective Transformer (in ms)	Speedup over XLA
100	28.31	20.27	1.40	16.03	1.77
200	54.47	40.08	1.36	30.15	1.81
300	80.53	59.11	1.36	41.27	1.95
400	106.5	78.38	1.36	54.12	1.97
500	132.35	98.03	1.37	65.92	2.01
1000	261.18	190.91	1.38	133.61	1.95

Source: https://github.com/bytedance/effective_transformer

Flash attention

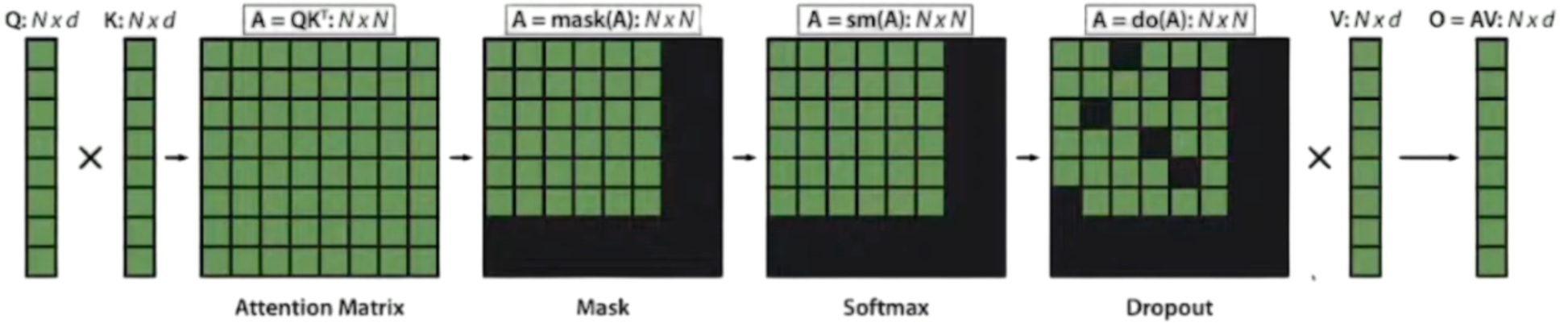
Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

Source: https://arxiv.org/abs/2205.14135

Flash attention

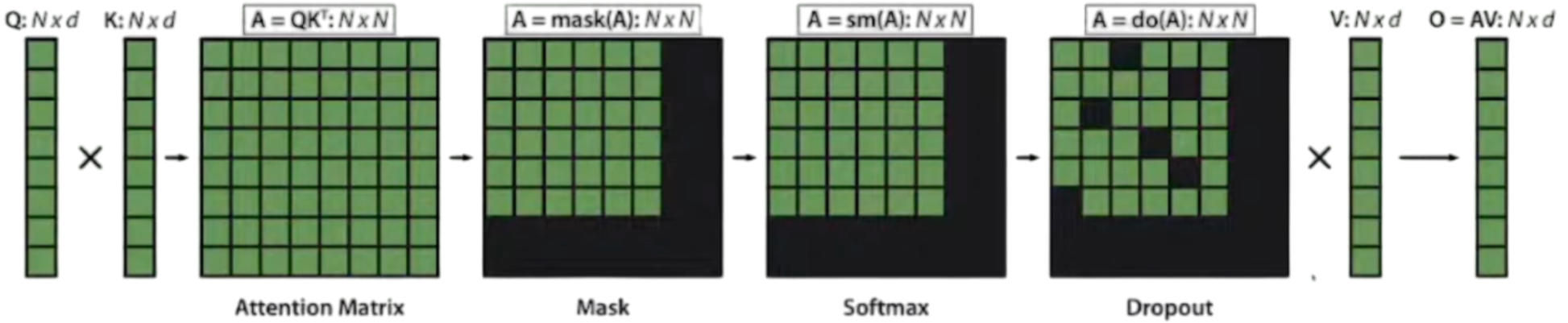


$$\mathbf{O} = \text{Dropout}(\text{Softmax}(\text{Mask}(\mathbf{QK}^\mathbf{T}))\mathbf{V}$$

Naive implementation requires repeated R/W from slow GPU HBM. Hard to scale to long sequences

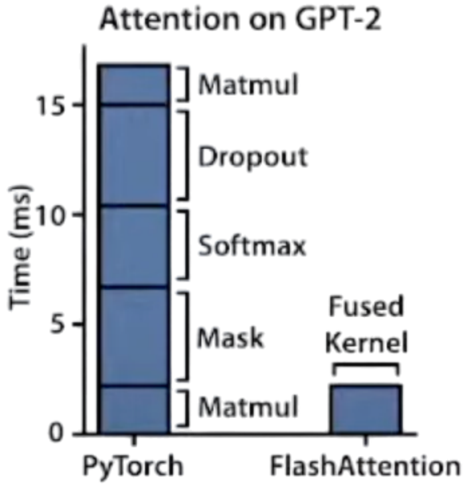
Source: https://arxiv.org/abs/2205.14135

Flash attention



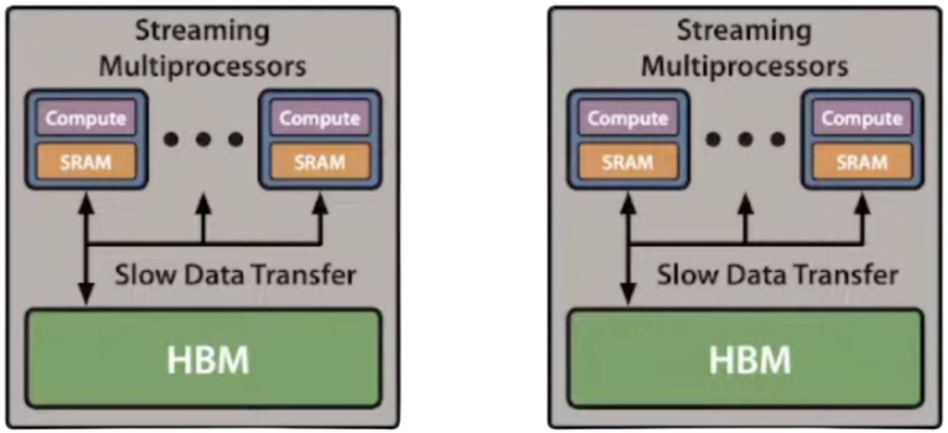
$$\mathbf{O} = \text{Dropout}(\text{Softmax}(\text{Mask}(\mathbf{QK}^\mathbf{T}))\mathbf{V}$$

Naive implementation requires repeated R/W from slow GPU HBM. Hard to scale to long sequences

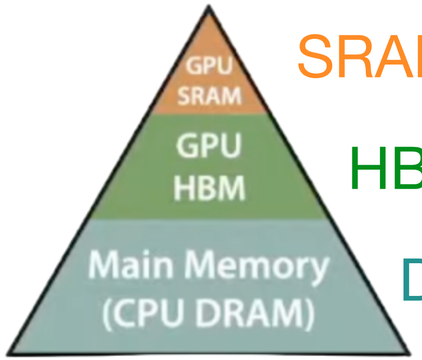


Source: https://arxiv.org/abs/2205.14135

Flash attention



GPU 0 … GPU N



SRAM 19TB/s (20MB)

HBM 1.5TB/s (40GB)

DRAM 13GB/s (> 1TB)

Source: https://arxiv.org/abs/2205.14135

Flash attention

Algorithm 1 FLASHATTENTION

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M.

1: Set block sizes $B_c = \left\lceil \frac{M}{4d} \right\rceil$, $B_r = \min\left(\left\lceil \frac{M}{4d} \right\rceil, d\right)$.
2: Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}$, $\ell = (0)_N \in \mathbb{R}^N$, $m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
3: Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \ldots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \ldots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \ldots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
4: Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \ldots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_i, \ldots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks $m_1, \ldots, m_{T_r}$ of size B_r each.
5: **for** $1 \le j \le T_c$ **do**
6: Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
7: **for** $1 \le i \le T_r$ **do**
8: Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
9: On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
10: On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
11: On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
12: Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1}(\mathrm{diag}(\ell_i) e^{m_i - m_i^{\mathrm{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.
13: Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.
14: **end for**
15: **end for**
16: Return $\mathbf{O}$.

Source: https://arxiv.org/abs/2205.14135

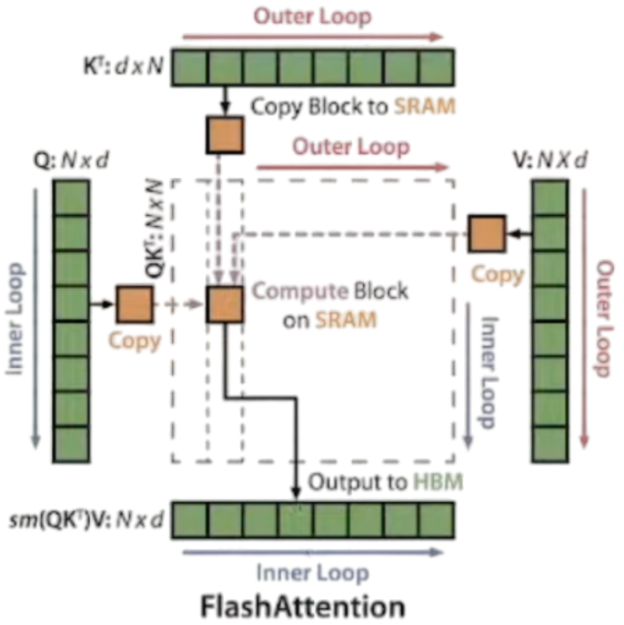
Flash attention

Decomposing large softmax into smaller ones by scaling.

$$\text{softmax}([A_1, A_2]) = [\alpha\ \text{softmax}(A_1), \beta\ \text{softmax}(A_2)]$$

$$\text{softmax}([A_1, A_2]) \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \alpha\ \text{softmax}(A_1)\ V_1 + \beta\ \text{softmax}(A_2) V_2$$

1. Load inputs by blocks from HBM to SRAM.
2. On chip, compute attention output wrt that block.
3. Update output in HBM by scaling.



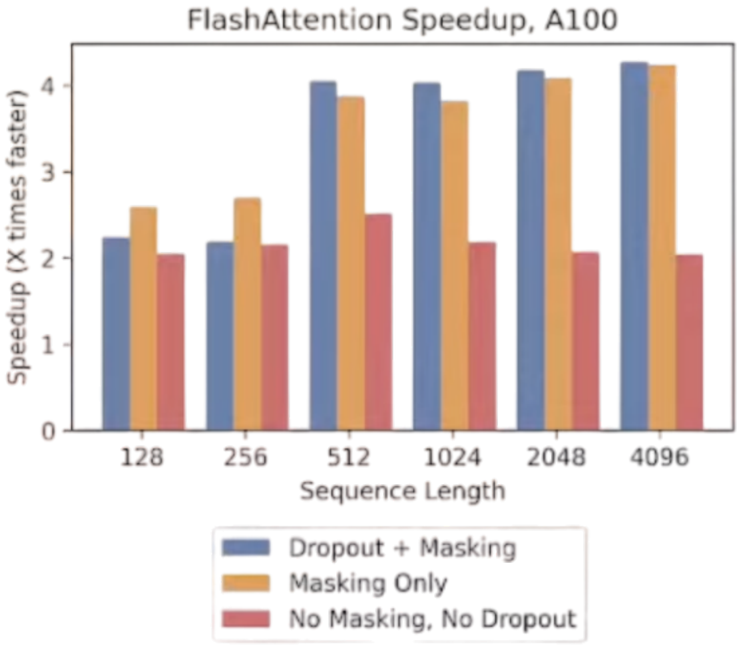
For vectors $x^{(1)}, x^{(2)} \in \mathbb{R}^B$, we can decompose the softmax of the concatenated $x = \left[x^{(1)}\ x^{(2)}\right] \in \mathbb{R}^{2B}$ as:

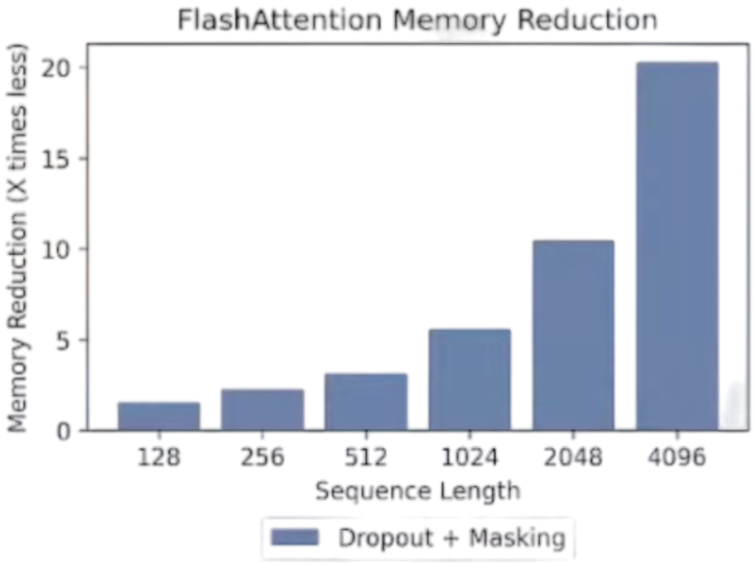
$$m(x) = m(\left[x^{(1)}\ x^{(2)}\right]) = \max(m(x^{(1)}), m(x^{(2)})), \quad f(x) = \left[e^{m(x^{(1)})-m(x)} f(x^{(1)}) \quad e^{m(x^{(2)})-m(x)} f(x^{(2)})\right],$$

$$\ell(x) = \ell(\left[x^{(1)}\ x^{(2)}\right]) = e^{m(x^{(1)})-m(x)} \ell(x^{(1)}) + e^{m(x^{(2)})-m(x)} \ell(x^{(2)}), \quad \text{softmax}(x) = \frac{f(x)}{\ell(x)}.$$

Source: https://arxiv.org/abs/2205.14135

Flash attention



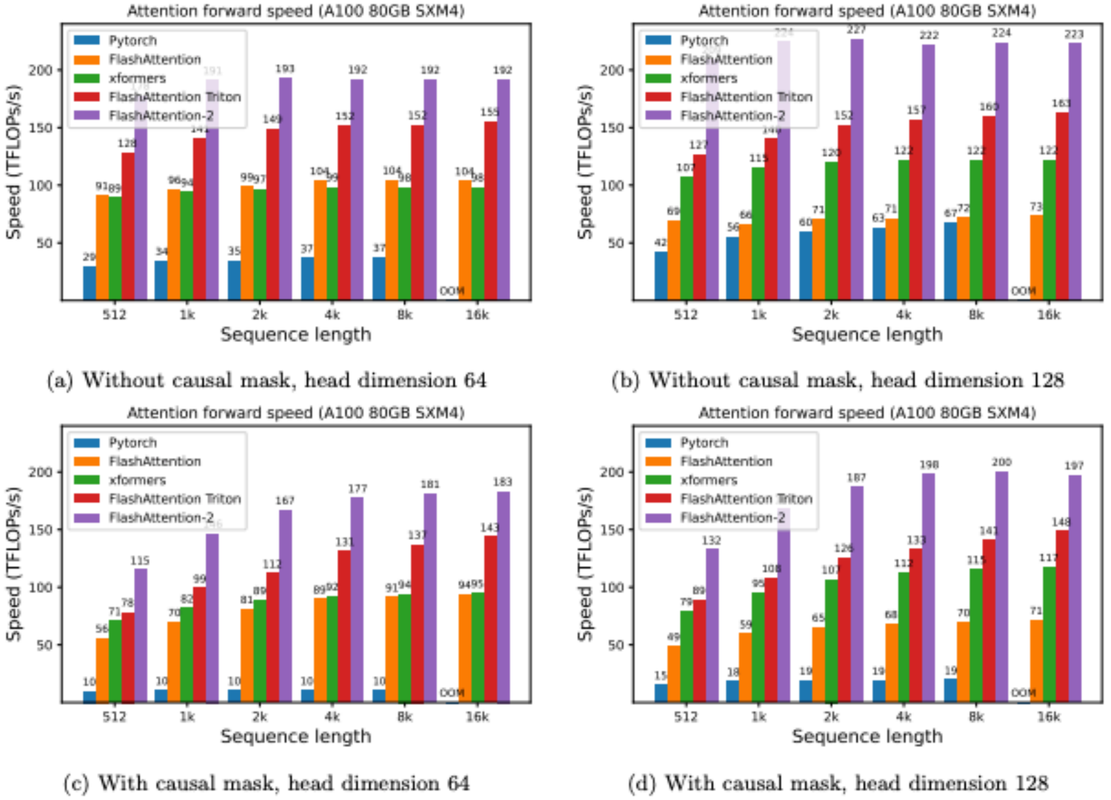


2-4x speedup — with no approximation!

10-20x memory reduction — memory linear in sequence length

Source: https://arxiv.org/abs/2205.14135

Flash attention 2



(a) Without causal mask, head dimension 64

(b) Without causal mask, head dimension 128

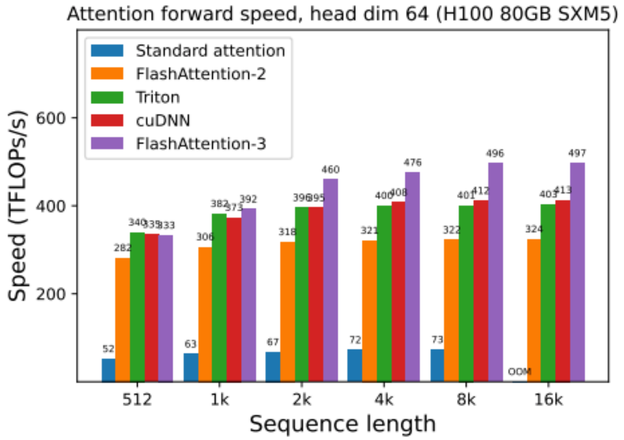
(c) With causal mask, head dimension 64

(d) With causal mask, head dimension 128

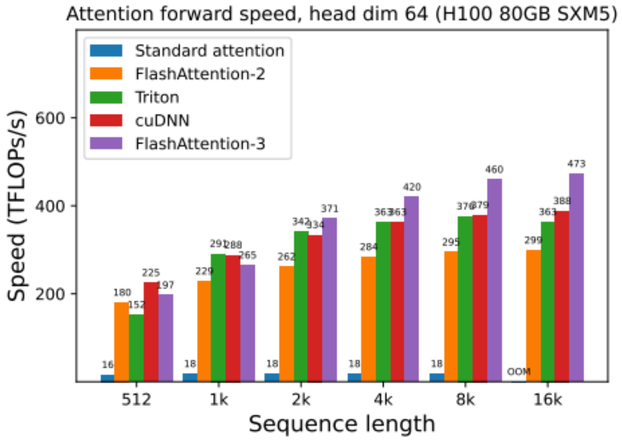
Figure 5: Attention forward speed on A100 GPU

Source: https://arxiv.org/pdf/2307.08691.pdf

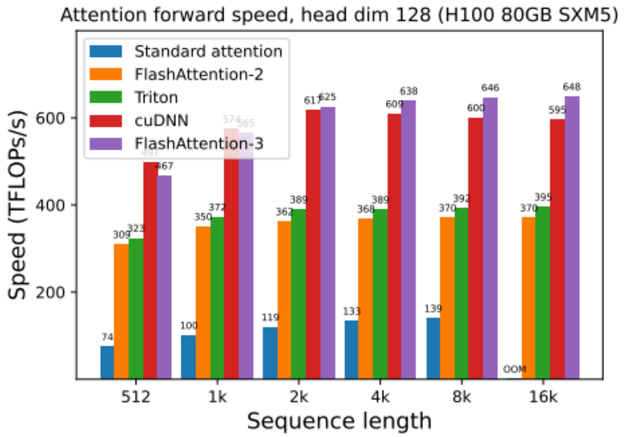
Flash attention 3



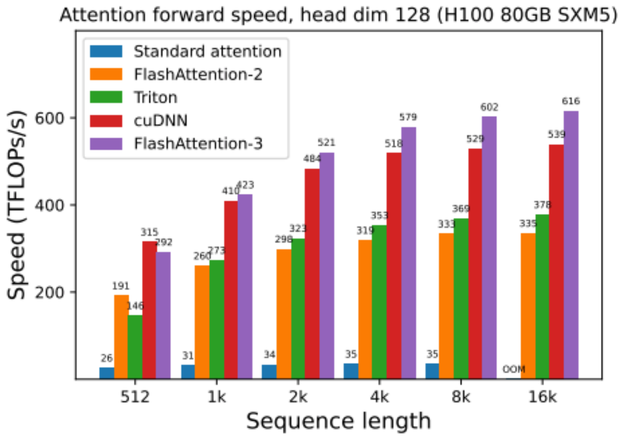
(a) Forward, without causal mask, head dim 64



(b) Forward, with causal mask, head dim 64



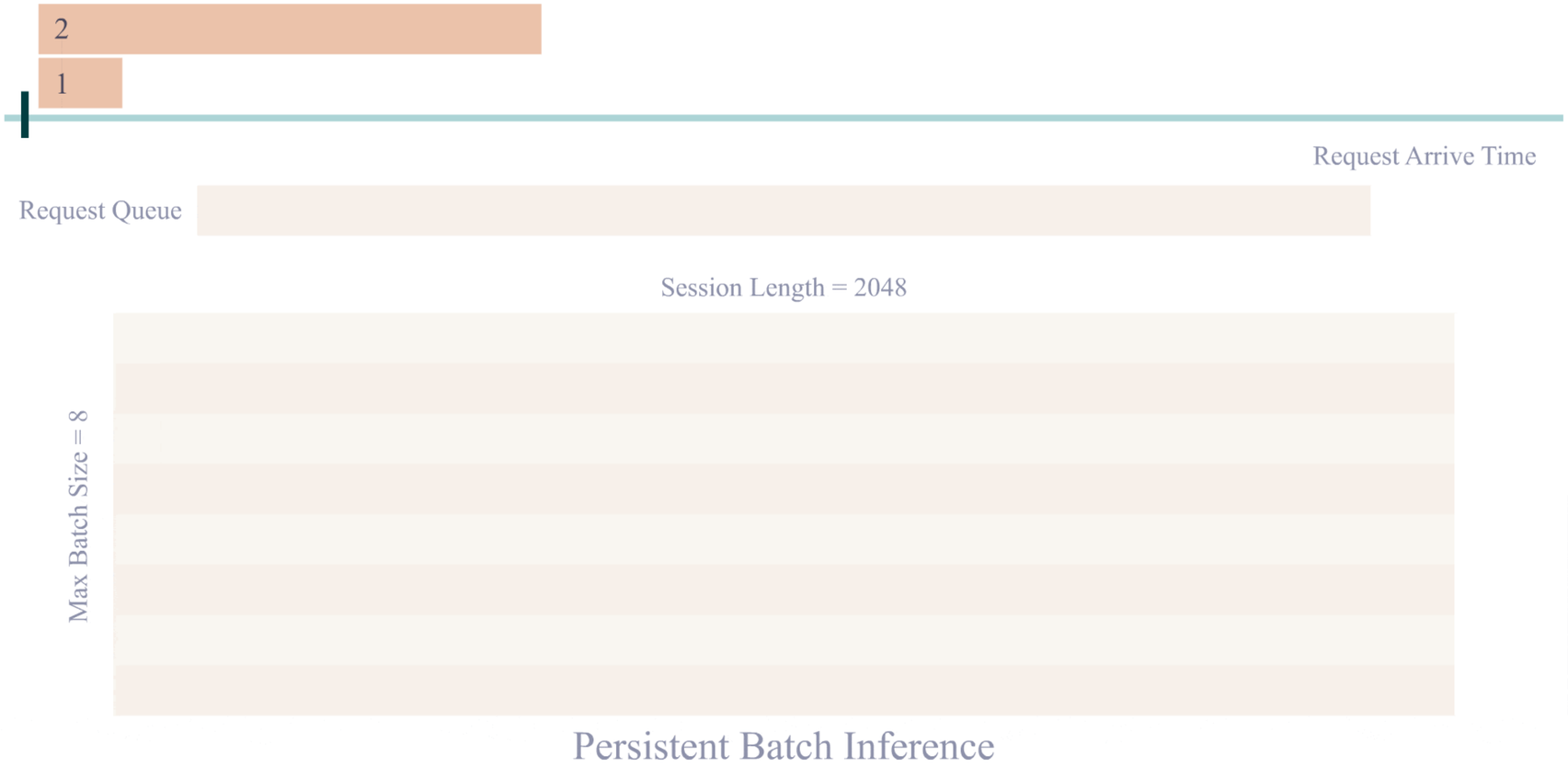
(c) Forward, without causal mask, head dim 128



(d) Forward, with causal mask, head dim 128

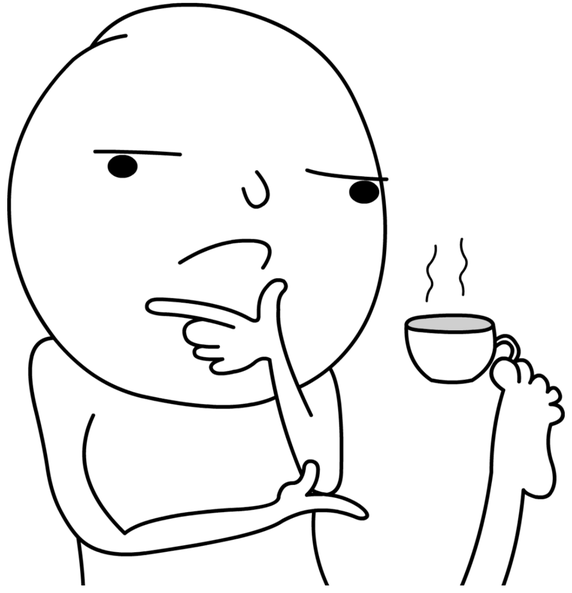
Source: https://arxiv.org/pdf/2407.08608

Continuous batch

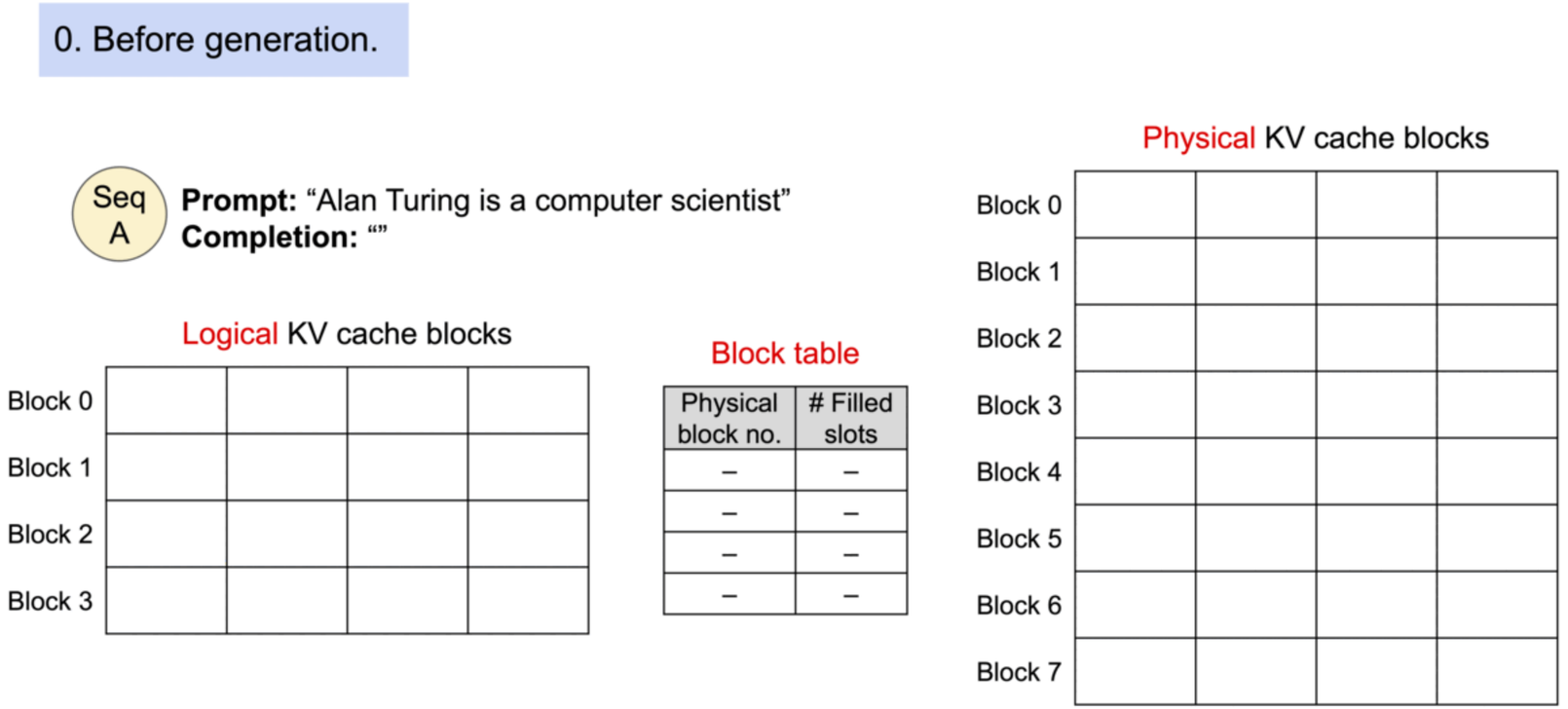


Source: https://github.com/InternLM/lmdeploy

What about long sequence batches?

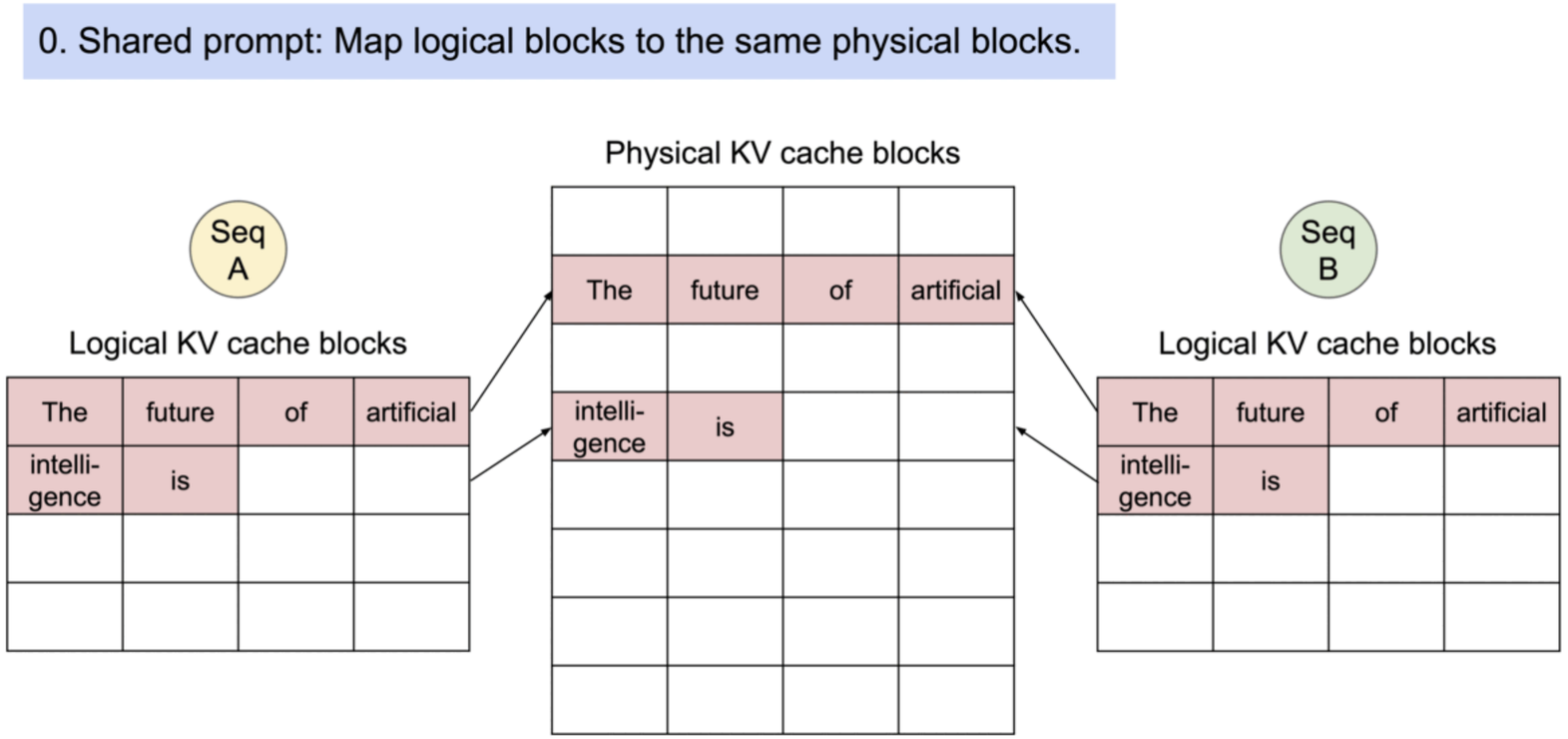


Paged Attention



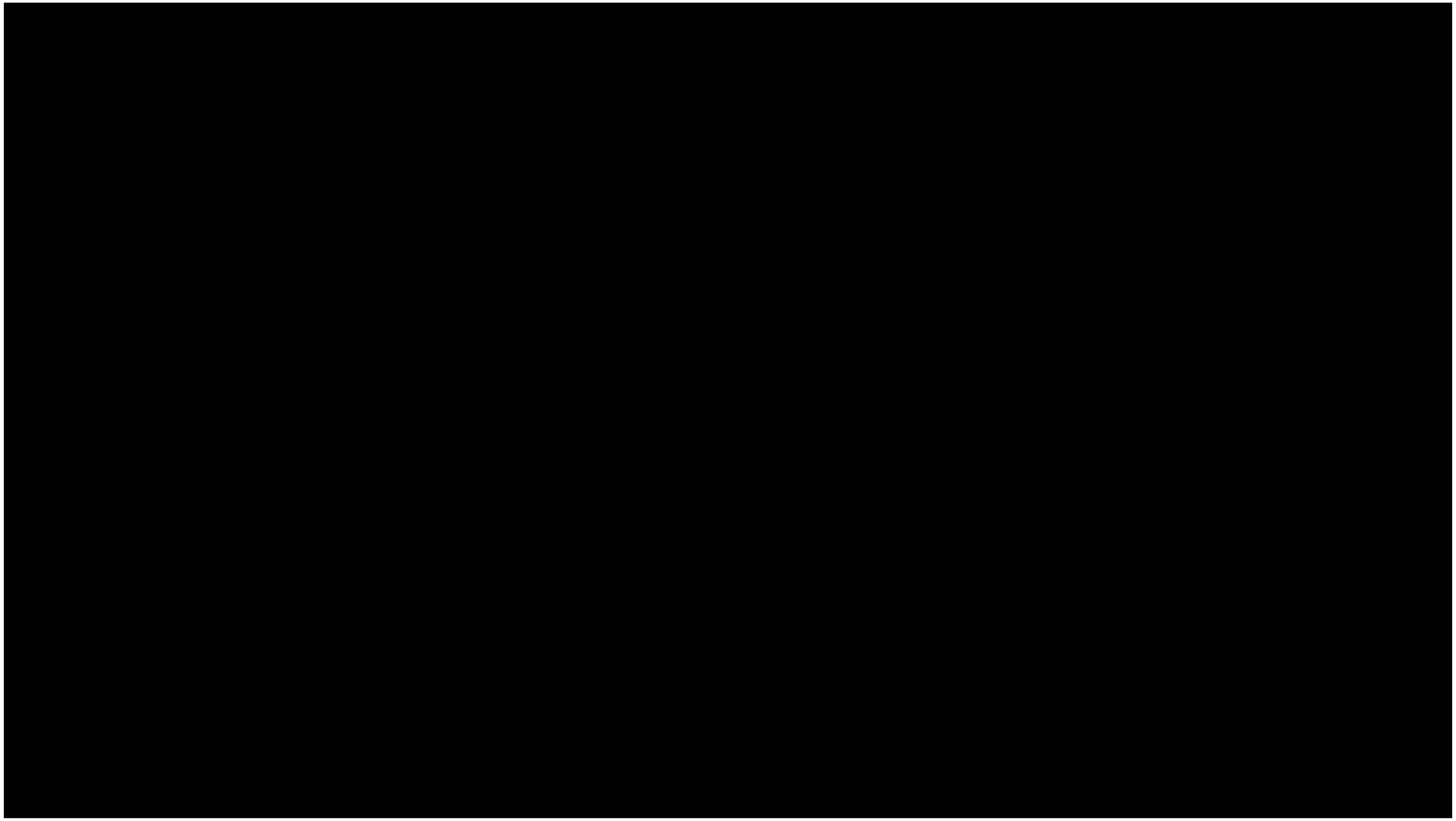
Source: https://blog.vllm.ai/2023/06/20/vllm.html

Paged Attention

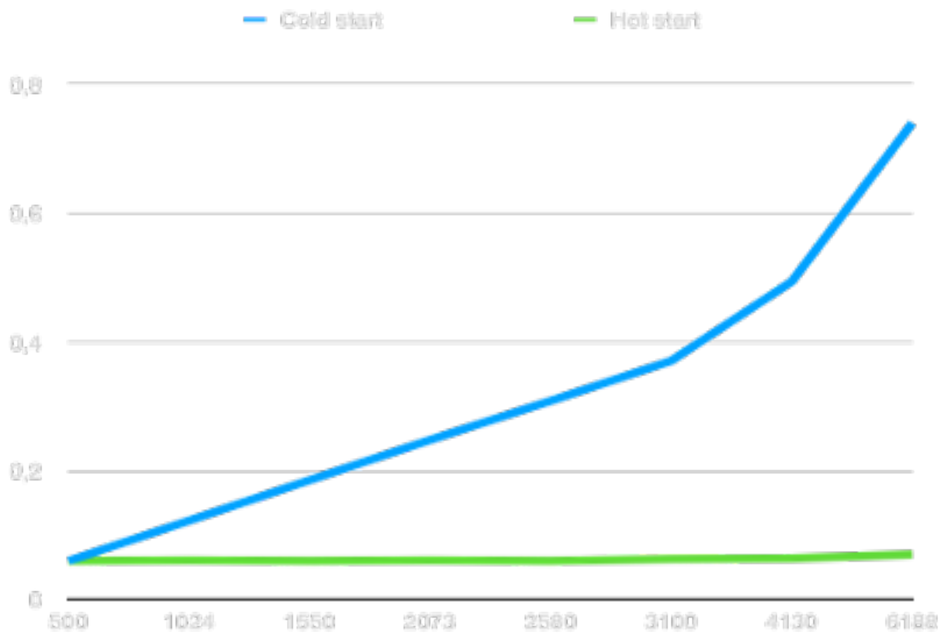


Source: https://blog.vllm.ai/2023/06/20/vllm.html

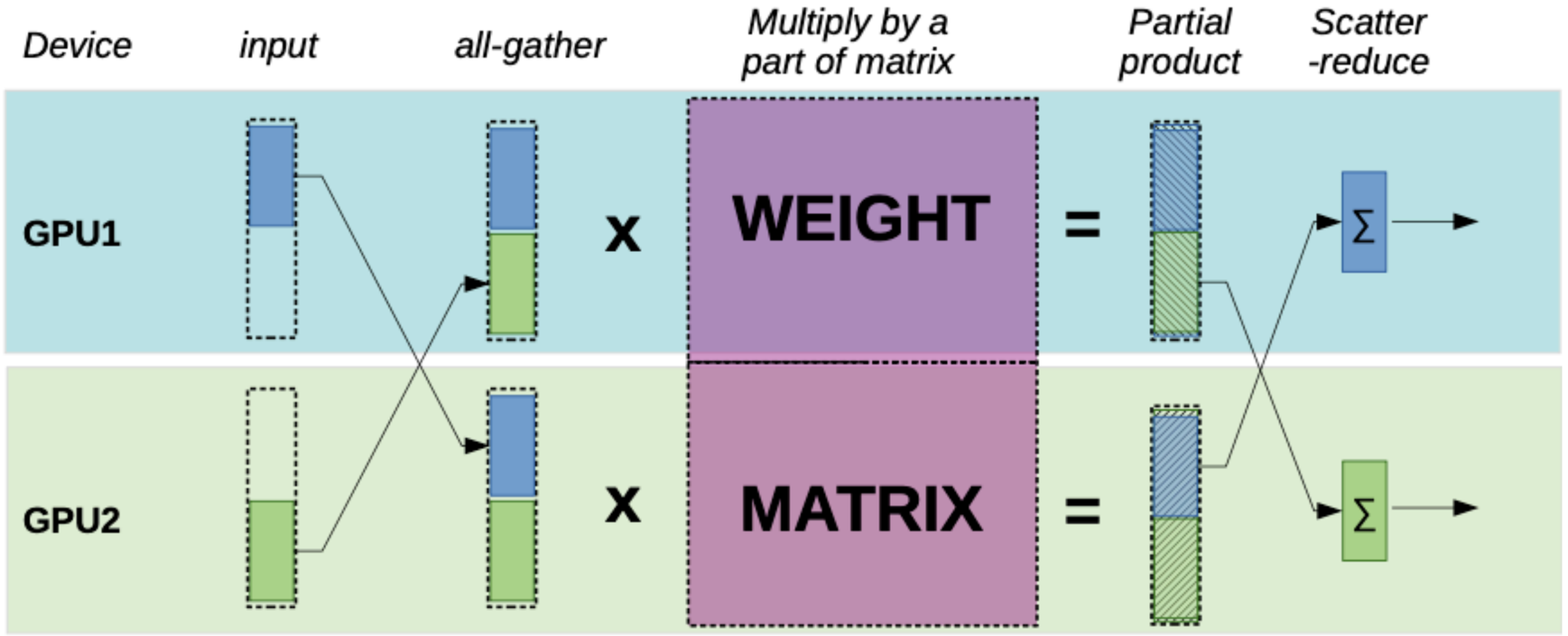
Huge KV caches



KV cache reuse



Tensor parallel



Tensor parallel

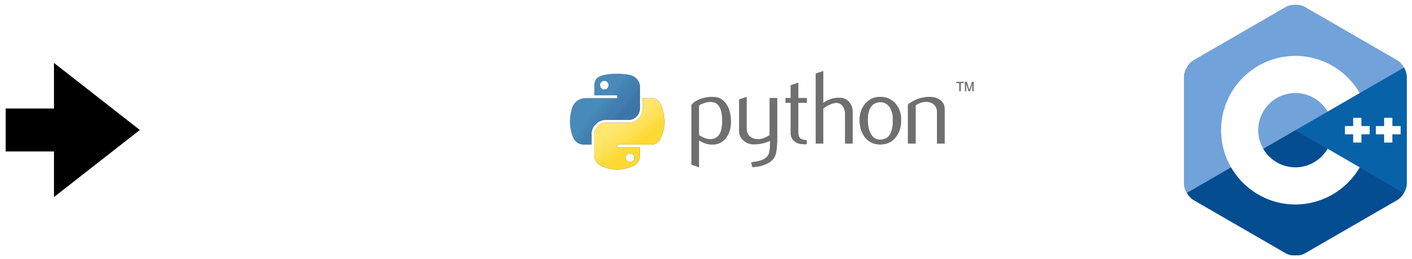
Performance of GPT-20B

Batch_size	Input Length	Output Length	Latency of single GPU (ms)	Latency of 2-way TP (ms)	Latency of 4-way TP (ms)	Latency of 8-way TP (ms)
1	20	8	225	147	102	89
2	20	8	225	152	108	94
4	20	8	228	158	113	100
8	20	8	239	169	121	107
16	20	8	268	191	133	113
32	20	8	331	230	155	127
64	20	8	452	314	200	169
128	20	8	726	484	318	256
256	20	8	1352	844	533	416
1	60	20	560	358	248	212
2	60	20	562	378	262	222
4	60	20	582	393	274	236
8	60	20	635	429	299	247
16	60	20	748	510	345	272
32	60	20	933	620	418	325
64	60	20	1352	887	574	454
128	60	20	2218	1384	928	699
256	60	20	4141	2424	1574	1152

Inference

- Prompt processing
- Autoregressive steps
- Hardware utilization

- Serving queries
- Business cases



Frameworks

https://github.com/vllm-project/vllm

https://github.com/sgl-project/sglang

https://github.com/NVIDIA/TensorRT-LLM

https://github.com/ggerganov/llama.cpp

Business

- Different scenarios
- Different sources of context
- Sampling control
  - Cycles
  - Images
  - Censorship

Current state