ЧТО ИСПОЛЬЗУЕМ

- Шрифт для заголовков – скачать
- Не в заголовках используем – Lab Grotesque K Regular (Основной текст)
- Основные цвета – ff007f 9700ff
- Дополнительные цвета темы можем использовать для графиков и сценариев, где нужно показать сравнение с 2-мя и более цветами

ЗАЧЕМ В ML НУЖНЫ ПРОИЗВОДНАЯ, GPU И АЛГОРИТМЫ

Александр Панкратов

Разработчик машинного обучения

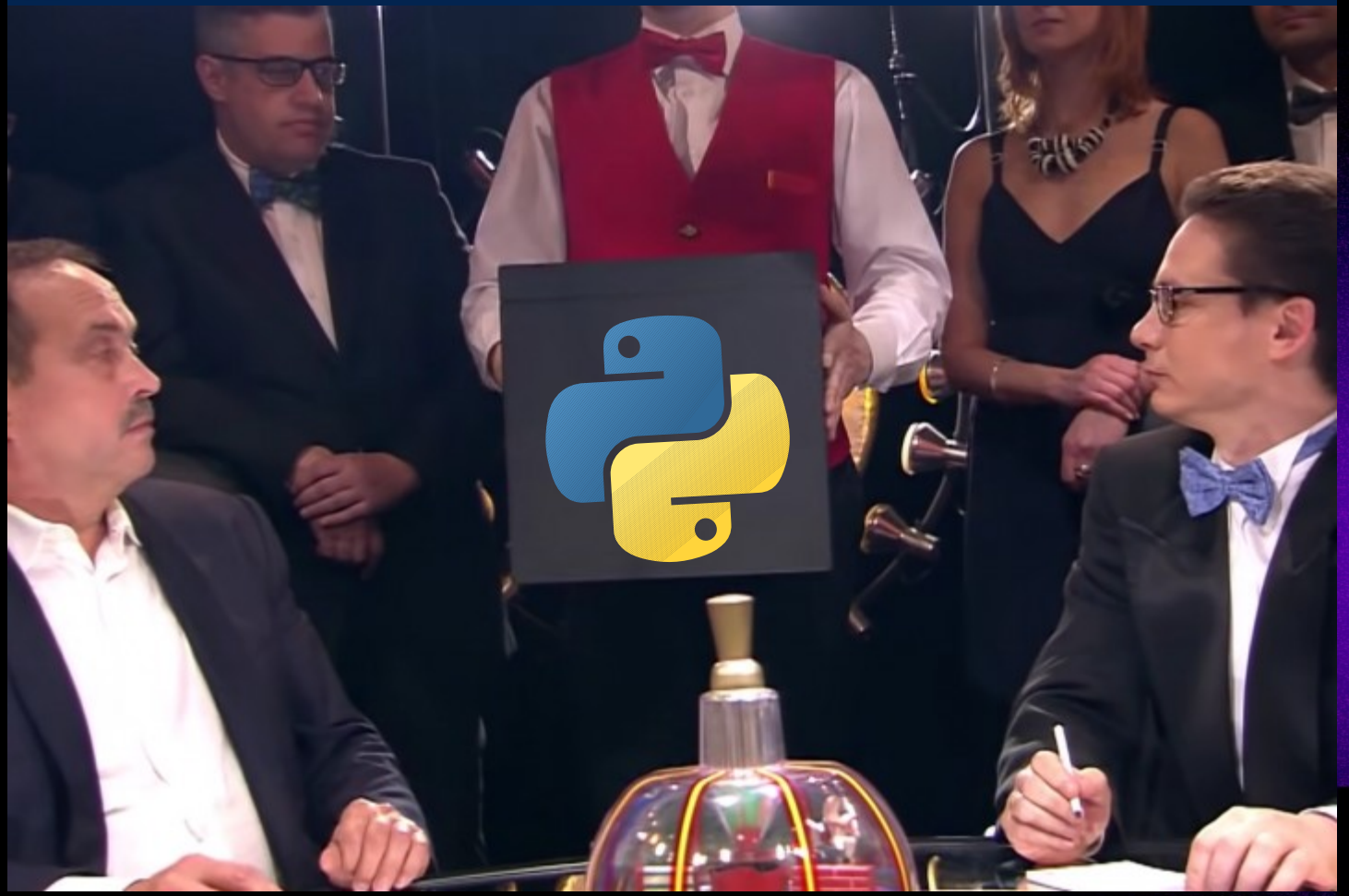
Что обсудим

- Для чего в точности нам нужны GPU, и не лучше ли Контуру купить чугунный мост вместо GPU-серверов?
- Что именно нужно проделать с производной, чтобы нейросеть обучалась.
- Где и зачем в МЛ приходится сесть и написать хороший код.

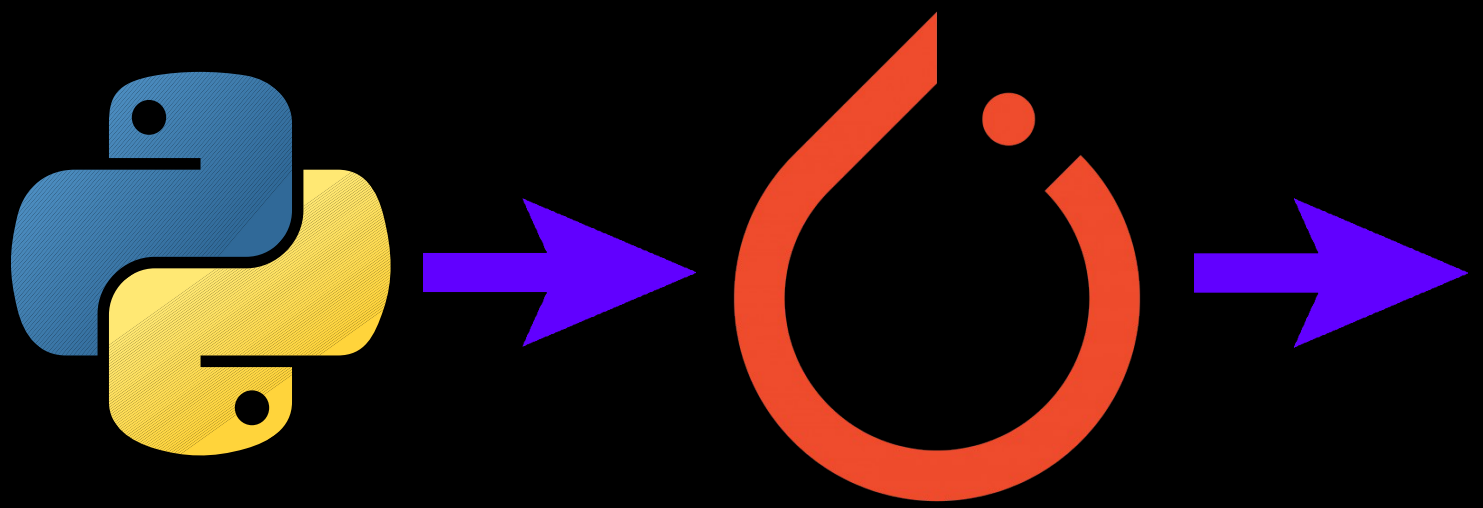
ЗАЧЕМ ДАТА-САТАНИСТУ GPU

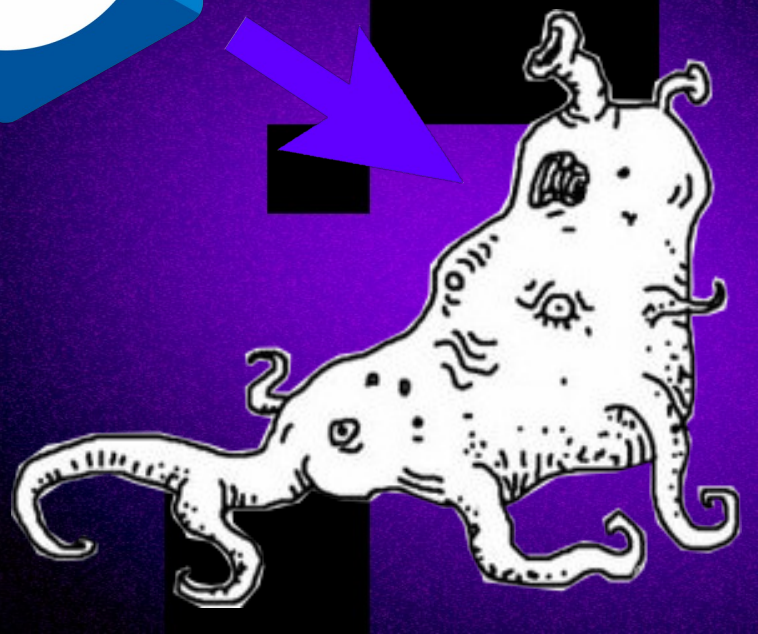
ЧТО ВНУТРИ НЕЙРОСЕТИ

```python
asr_model = nemo_asr.models.EncDecCTCModelBPE.from_pretrained(
    model_name="stt_ru_conformer_ctc_large",
)
```



ЧТО ВНУТРИ НЕЙРОСЕТИ





Robert_Crovella Moderator

It's probably coming from a cublas call, or a library that uses cublas, like cudnn. You won't find these kernels documented anywhere.

СКАЛЬПЕЛЬ И МИКРОСКОП

```
model, data = get_model_and_data(model_name="stt_ru_conformer_ctc_large",device='cpu')

with profiler.profile(with_stack=True, activities=[profiler.ProfilerActivity.CPU]) as prof:
    with torch.no_grad():
        for processed_signal, processed_signal_length in data:
            model.encoder(audio_signal=processed_signal, length=processed_signal_length)

with open('etc/profiler_outputs/cpu_forward_profiling', mode='w') as out:
    out.write(prof.key_averages().table(sort_by='self_cpu_time_total'))
```

ЧТО ПРОИСХОДИТ КОГДА НЕТ GPU

CPU

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
aten::addmm	45.62%	2.335s	48.04%	2.458s	264.888us	9280
aten::mkldnn_convolution	23.35%	1.195s	23.67%	1.211s	362.648us	3340
aten::copy_	4.78%	244.852ms	4.78%	244.852ms	11.193us	21876
aten::mm	3.36%	171.688ms	3.36%	171.767ms	149.103us	1152
aten::bmm	3.11%	158.910ms	3.11%	159.221ms	46.071us	3456
aten::add	2.82%	144.425ms	2.84%	145.091ms	17.439us	8320
aten::silu	2.08%	106.633ms	2.08%	106.633ms	30.855us	3456
aten::native_layer_norm	1.64%	83.711ms	1.98%	101.436ms	17.610us	5760
aten::_slow_conv2d_forward	1.07%	54.978ms	1.22%	62.574ms	256.452us	244

ЧТО ПРОИСХОДИТ КОГДА ЕСТЬ GPU

CPU

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
aten::addmm	45.62%	2.335s	48.04%	2.458s	264.888us	9280
aten::mkldnn_convolution	23.35%	1.195s	23.67%	1.211s	362.648us	3340
aten::copy_	4.78%	244.852ms	4.78%	244.852ms	11.193us	21876
aten::mm	3.36%	171.688ms	3.36%	171.767ms	149.103us	1152
aten::bmm	3.11%	158.910ms	3.11%	159.221ms	46.071us	3456

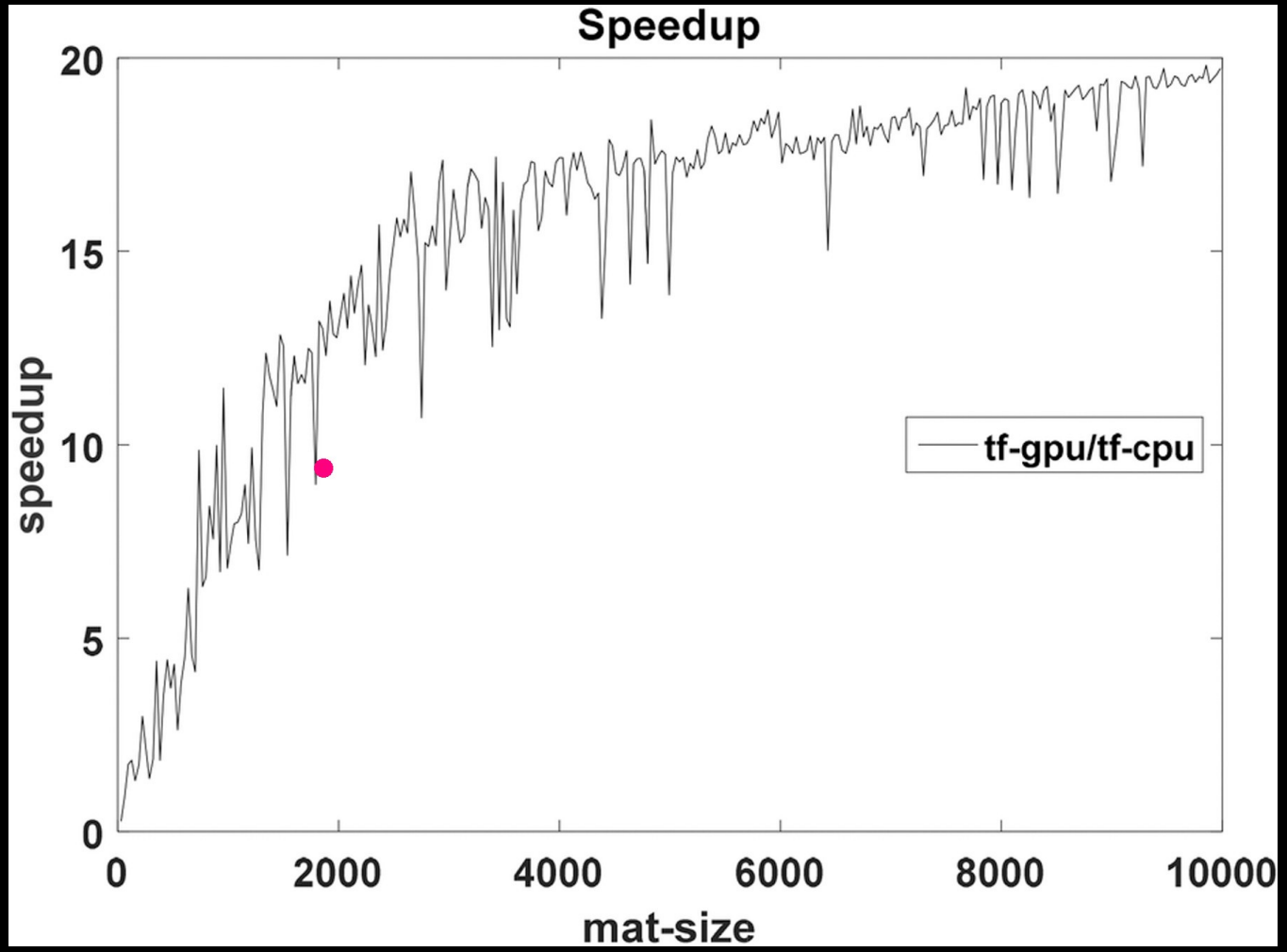
GPU

Name	Self CPU %	Self CPU	CPU time avg	Self CUDA	Self CUDA %	CUDA time avg	# of Calls
aten::addmm	10.54%	148.755ms	25.925us	146.863ms	42.41%	15.826us	9280
ampere_sgemm_64x32_sliced1x4_tn	0.00%	0.000us	0.000us	93.975ms	27.14%	13.448us	6988
aten::cudnn_convolution	3.75%	52.891ms	33.350us	51.405ms	14.84%	21.137us	2432
aten::bmm	3.38%	47.668ms	18.970us	40.111ms	11.58%	11.606us	3456
ampere_sgemm_128x128_tn	0.00%	0.000us	0.000us	25.050ms	7.23%	10.872us	2304
ampere_sgemm_128x32_tn	0.00%	0.000us	0.000us	21.646ms	6.25%	22.269us	972
ampere_sgemm_32x32_sliced1x4_tn	0.00%	0.000us	0.000us	20.528ms	5.93%	10.093us	2034

2235 / (148.755 + 146.863) ≈ 7.56

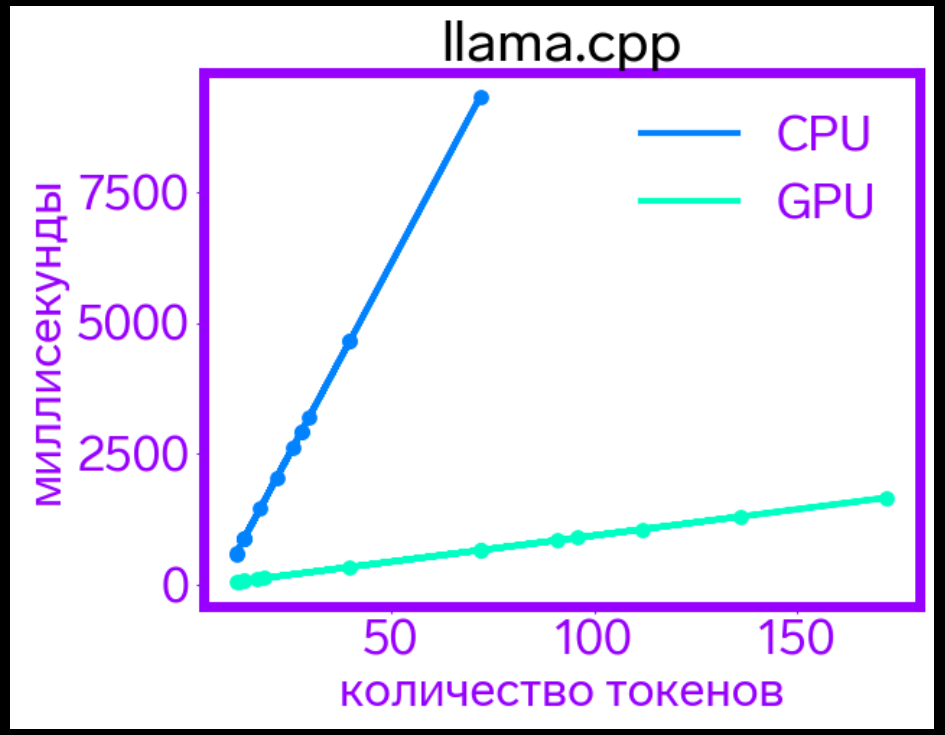
1195 / 3340*2432 / (52.891 + 51.405) ≈ 8.34

ЧТО ПРОИСХОДИТ КОГДА ЕСТЬ GPU



https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/joe.2018.9178

ЧТО ПРОИСХОДИТ КОГДА ЕСТЬ GPU



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

С GPU быстрее в 10-14 раз

GPU v CPU (single precision, batch size one)

Device	Framework	Seconds to Finish
NVIDIA A100 80GB PCIe	pytorch	6.49
Quadro RTX 8000	pytorch	12.3
AMD EPYC 7352 24-Core Processor	onnx	223.19
Intel(R) Core(TM) i7-6850K CPU @ 3.60GHz	onnx	286.13
Intel(R) Core(TM) i7-6850K CPU @ 3.60GHz	pytorch	458.97
AMD EPYC 7352 24-Core Processor	pytorch	529.93

lambdalabs.com/blog/inference-benchmark-stable-diffusion

С GPU быстрее в 17-35 раз


почему

КТО ТАКОЙ ADDMM

torch.addmm

torch.addmm(*input*, *mat1*, *mat2*, ***, *beta=1*, *alpha=1*, *out=None*) → Tensor

Performs a matrix multiplication of the matrices `mat1` and `mat2`. The matrix `input` is added to the final result.

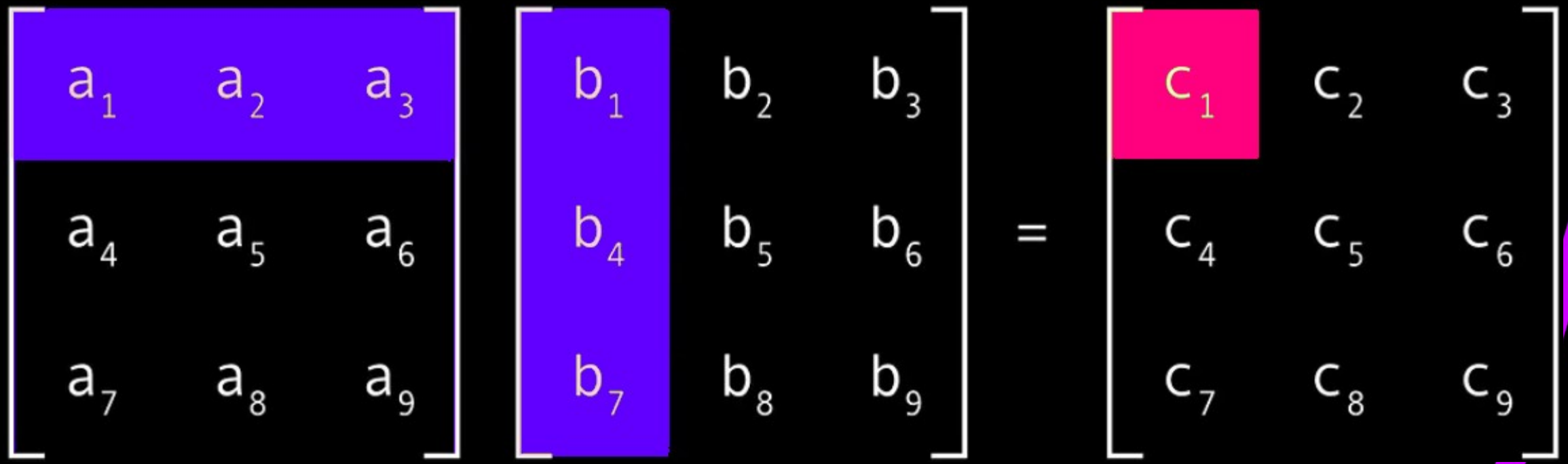
If `mat1` is a $(n \times m)$ tensor, `mat2` is a $(m \times p)$ tensor, then `input` must be broadcastable with a $(n \times p)$ tensor and `out` will be a $(n \times p)$ tensor.

`alpha` and `beta` are scaling factors on matrix-vector product between `mat1` and `mat2` and the added matrix `input` respectively.

$$\text{out} = \beta\ \text{input} + \alpha\ (\text{mat1}_i \mathbin{@} \text{mat2}_i)$$

https://pytorch.org/docs/2.4/generated/torch.addmm.html

КАК УМНОЖАТЬ МАТРИЦЫ



$$c_1 = a_1 b_1 + a_2 b_4 + a_3 b_7$$

https://charchithowitzer.medium.com/matrix-multiplication-why-is-it-a-big-deal-cc8ef7490008

КТО ТАКОЙ CONVOLUTION

Conv1d

CLASS torch.nn.Conv1d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

Applies a 1D convolution over an input signal composed of several input planes.

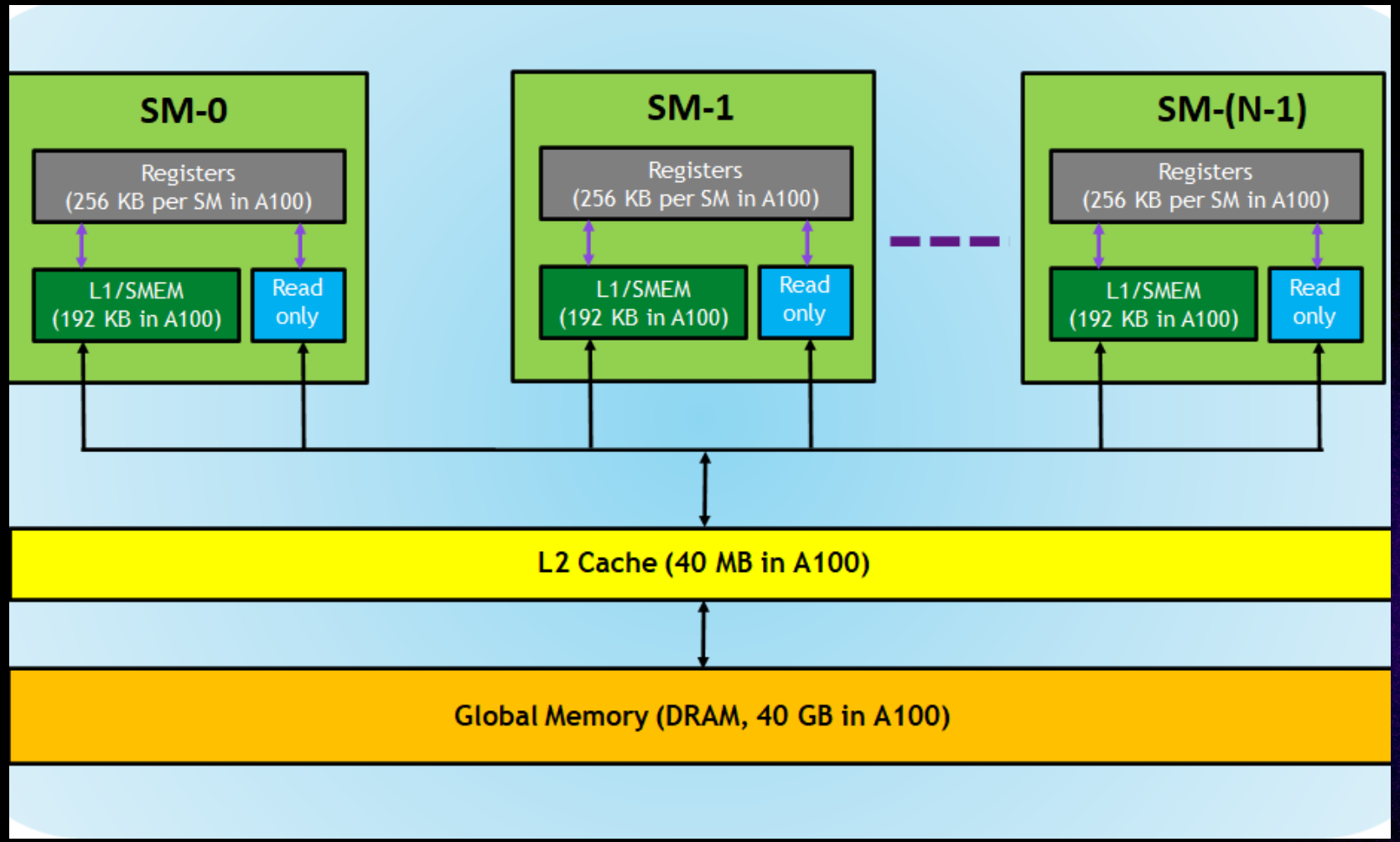
In the simplest case, the output value of the layer with input size (N, C_{in}, L) and output $(N, C_{\text{out}}, L_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

https://pytorch.org/docs/2.4/generated/torch.nn.Conv1d.html#torch.nn.Conv1d


почему

ВНУТРИ GPU



https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model/

КТО ТАКОЙ STREAMING MULTIPROCESSOR?

16.7. Compute Capability 8.x

16.7.1. Architecture

A Streaming Multiprocessor (SM) consists of:

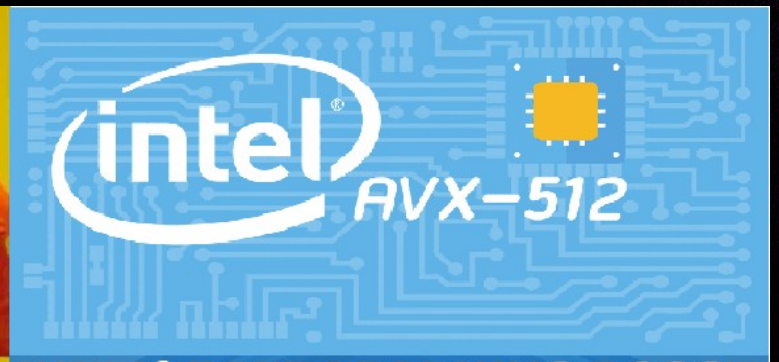
- 64 FP32 cores for single-precision arithmetic operations in devices of compute capability 8.0 and 128 FP32 cores in devices of compute capability 8.6, 8.7 and 8.9,
- 32 FP64 cores for double-precision arithmetic operations in devices of compute capability 8.0 and 2 FP64 cores in devices of compute capability 8.6, 8.7 and 8.9
- 64 INT32 cores for integer math,
- 4 mixed-precision Third-Generation Tensor Cores supporting half-precision (fp16), `__nv_bfloat16`, `tf32`, sub-byte and double precision (fp64) matrix arithmetic for compute capabilities 8.0, 8.6 and 8.7 (see Warp matrix functions for details),
- 4 mixed-precision Fourth-Generation Tensor Cores supporting `fp8`, `fp16`, `__nv_bfloat16`, `tf32`, sub-byte and `fp64` for compute capability 8.9 (see Warp matrix functions for details),
- 16 special function units for single-precision floating-point transcendental functions,
- 4 warp schedulers.

Overall, H100 provides approximately 6x compute performance improvement over A100 when factoring in all the new compute technology advances in H100. Figure 10 summarizes the improvements in H100 in a cascading manner:

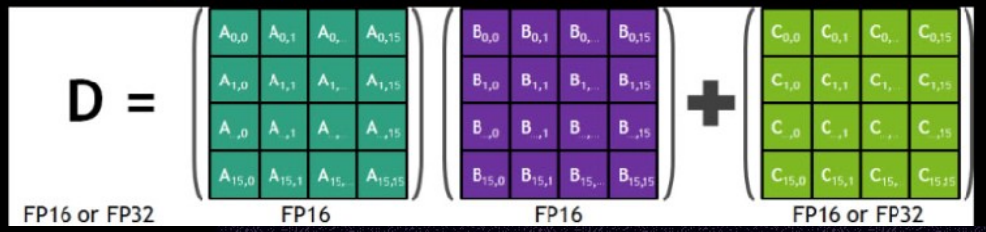
- 132 SMs provide a 22% SM count increase over the A100 108 SMs

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#architecture-7-x

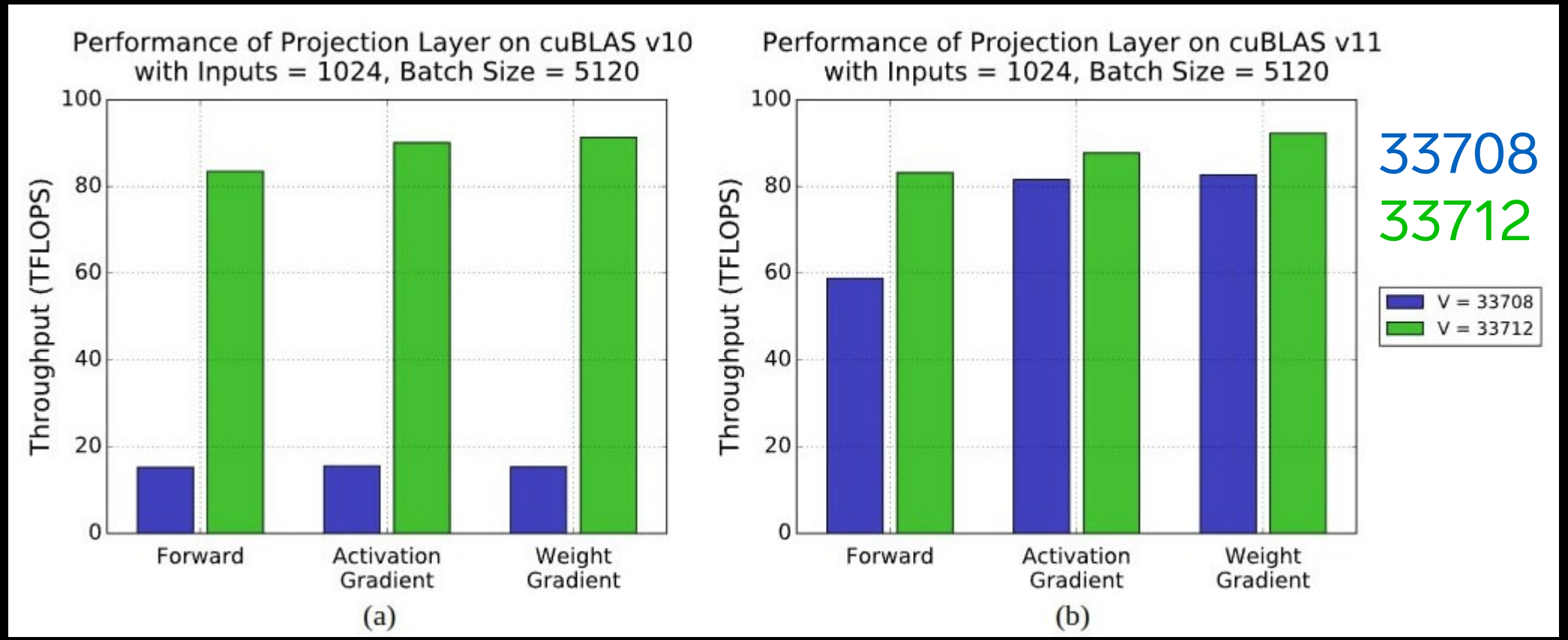
ДАВАЙТЕ ПРОСТО КУПИМ БОЛЬШЕ CPU



Volta was also the first architecture to introduce a brand new category of execution unit, the tensor core. This hardware is meant to accelerate matrix multiplication for deep learning workloads and is *incredibly * fast. Each tensor core can execute an entire 16x16 mma (matrix multiply and accumulate) in a single clock cycle (~0.75ns for RTX 3090).



ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ СЛОЖНЫ



https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ СЛОЖНЫ

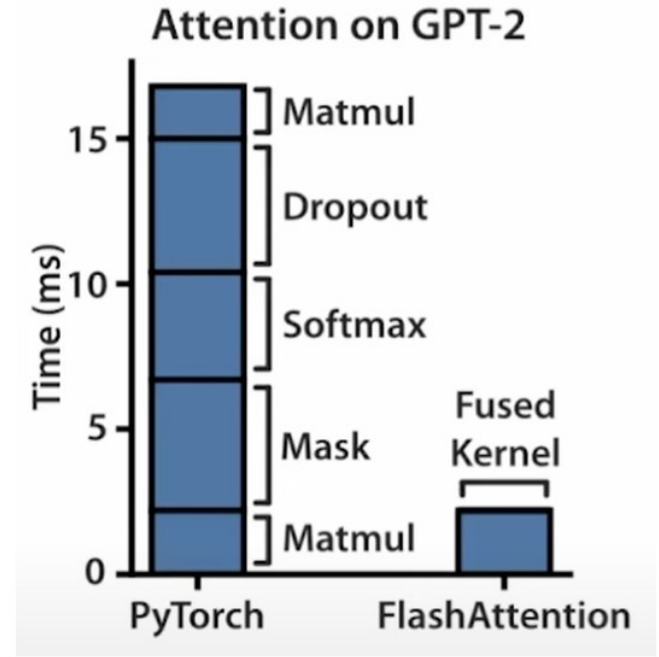
Andrej Karpathy 

@karpathy

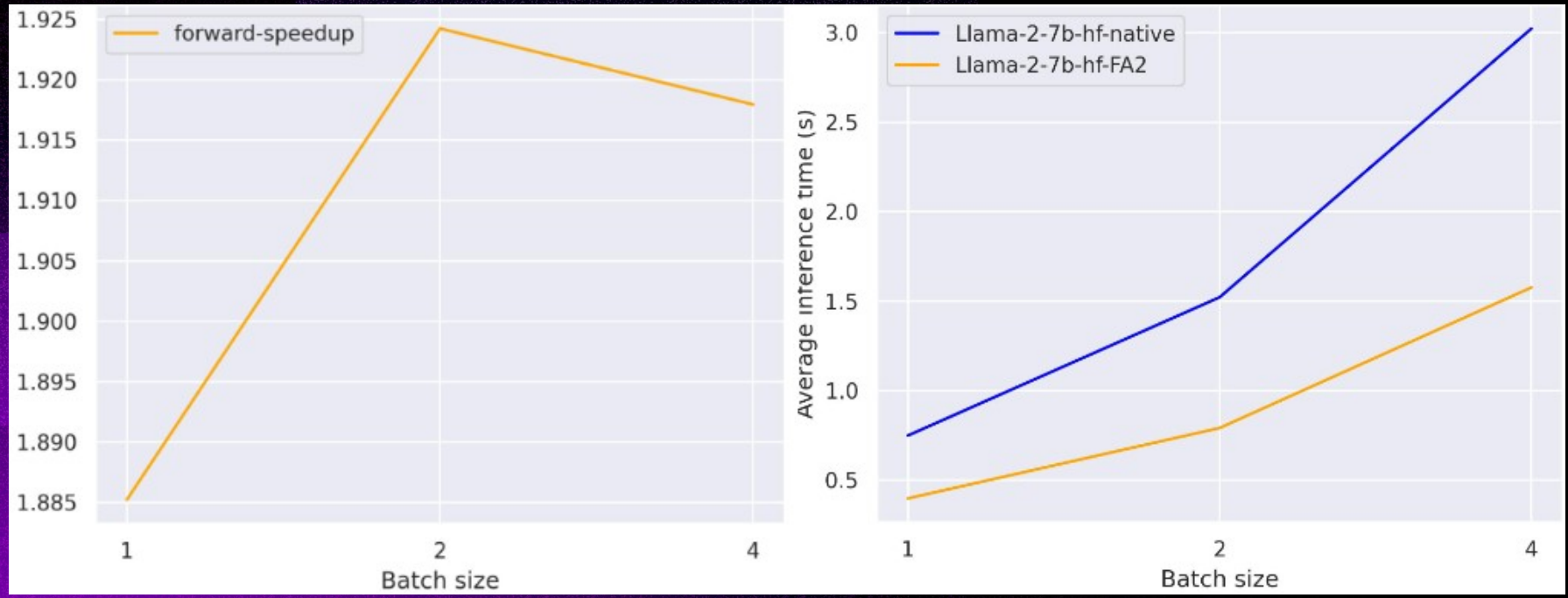
The most dramatic optimization to nanoGPT so far (~25% speedup) is to simply increase vocab size from 50257 to 50304 (nearest multiple of 64). This calculates added useless dimensions but goes down a different kernel path with much higher occupancy. Careful with your Powers of 2.

https://x.com/karpathy/status/1621578354024677377

НО ЕСЛИ СМОЧЬ, ТО...



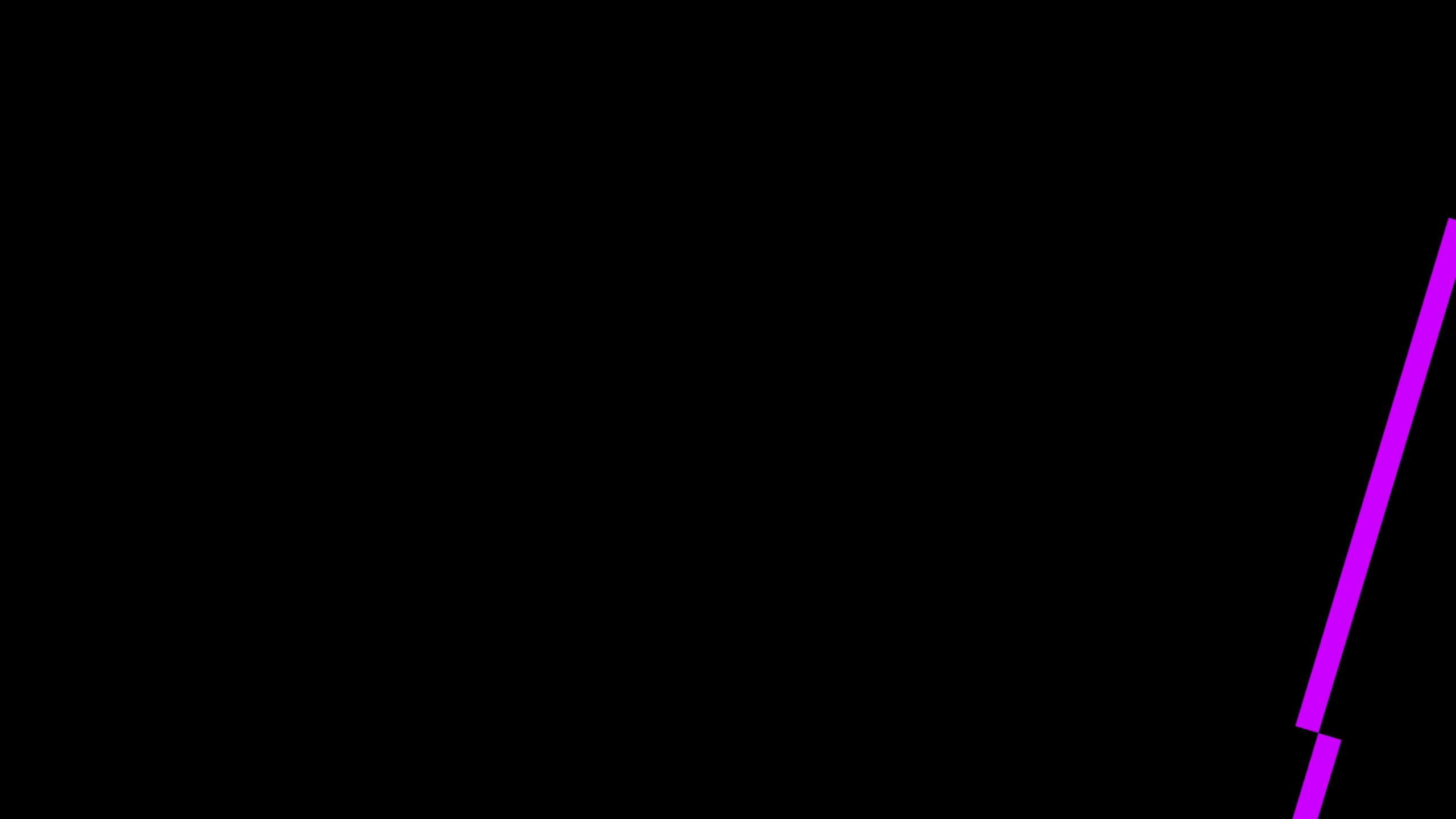
arxiv.org/abs/2205.14135

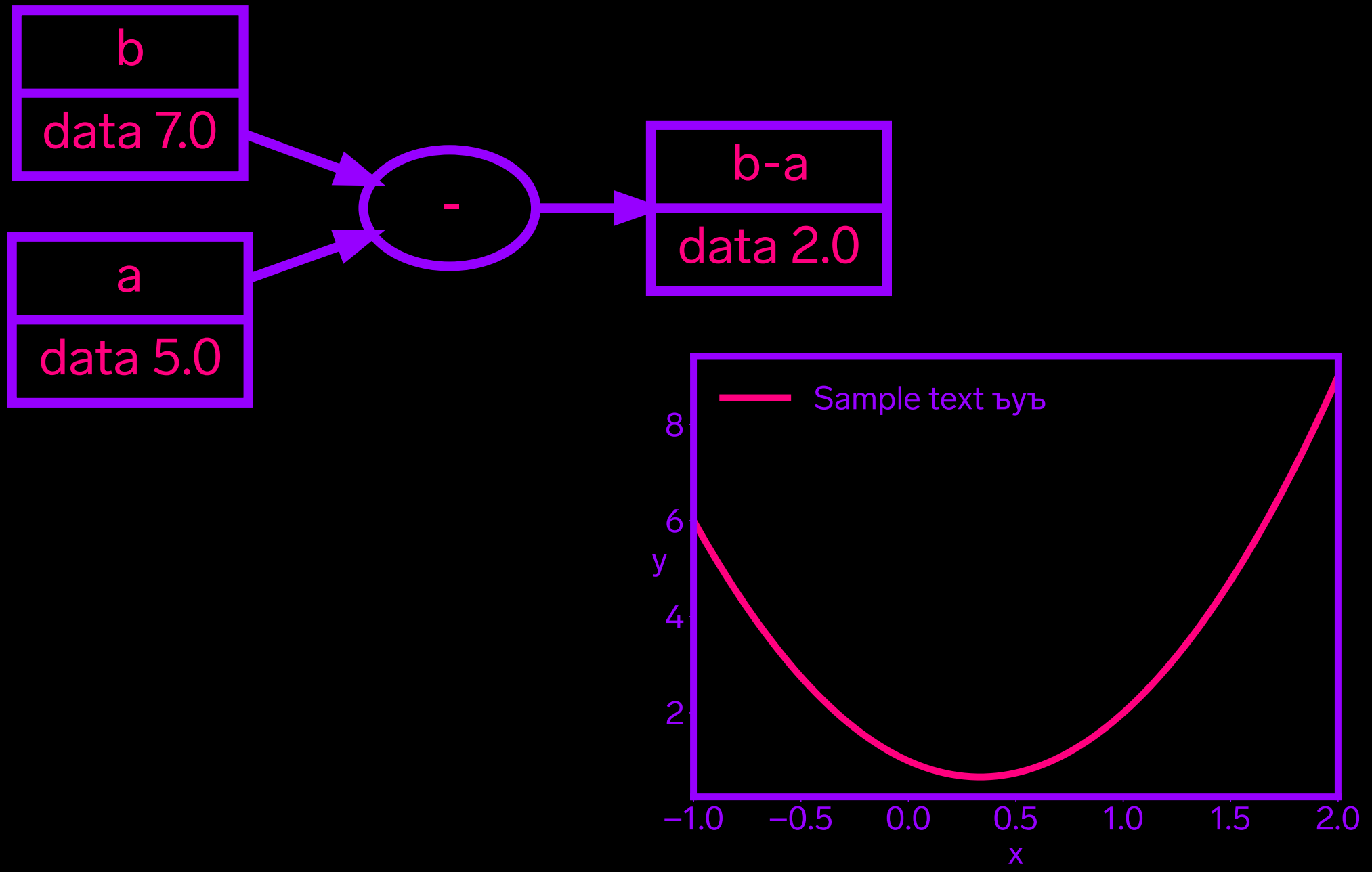


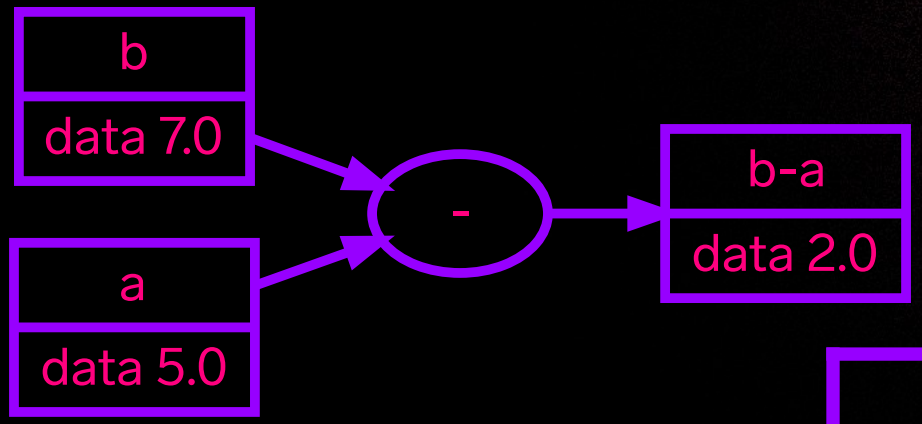
huggingface.co/docs/transformers/en/perf_infer_gpu_one

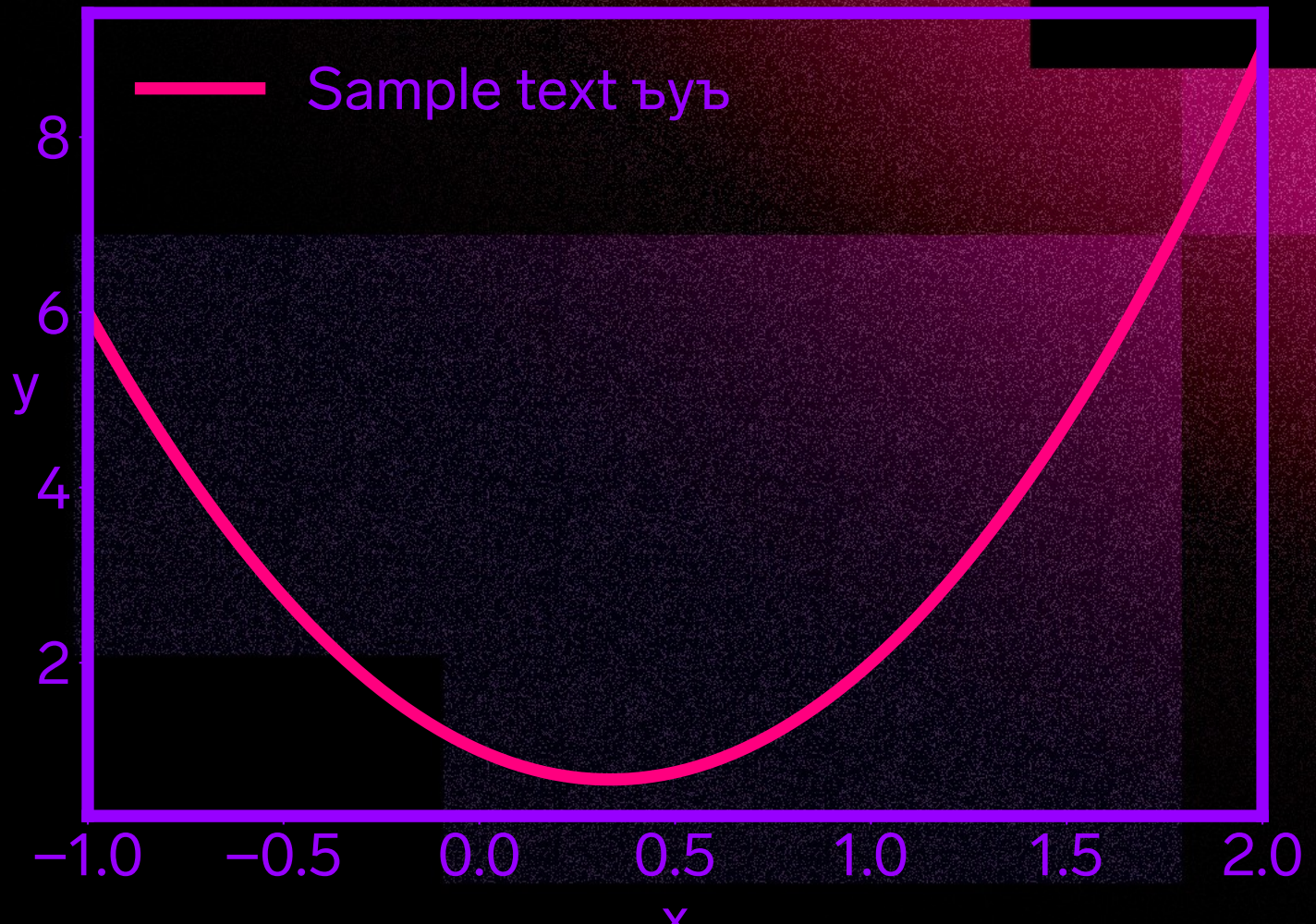
ПРОИЗВОДНАЯ И ЕЁ ДРУЗЬЯ

ВСПОМНИТЬ ВСЁ




b
data 7.0
a
data 5.0
-
b-a
data 2.0
Sample text ъуъ
8
6
4
2
y
–1.0
–0.5
0.0
0.5
1.0
1.5
2.0
x


b
data 7.0
a
data 5.0
-
b-a
data 2.0


Sample text ъуъ
8
6
4
2
y
–1.0
–0.5
0.0
0.5
1.0
1.5
2.0
x

А ЧТО ЕСЛИ ДАЖЕ GPU НЕ ХВАТАЕТ?