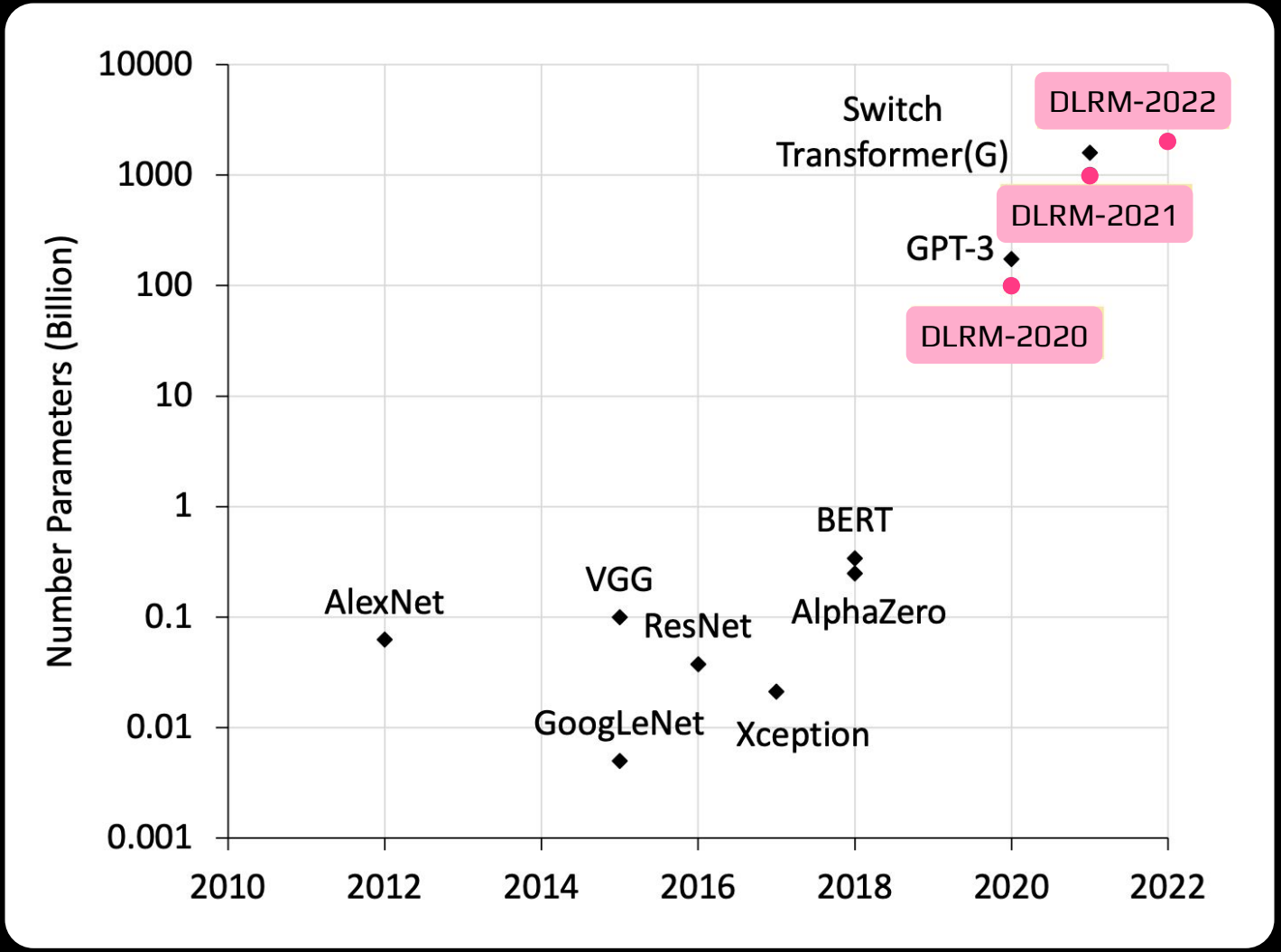
AI VK

Проблемы обучения и инференса больших табличных нейросетей

Болозовский Роман, ML-инженер

О чём будет доклад

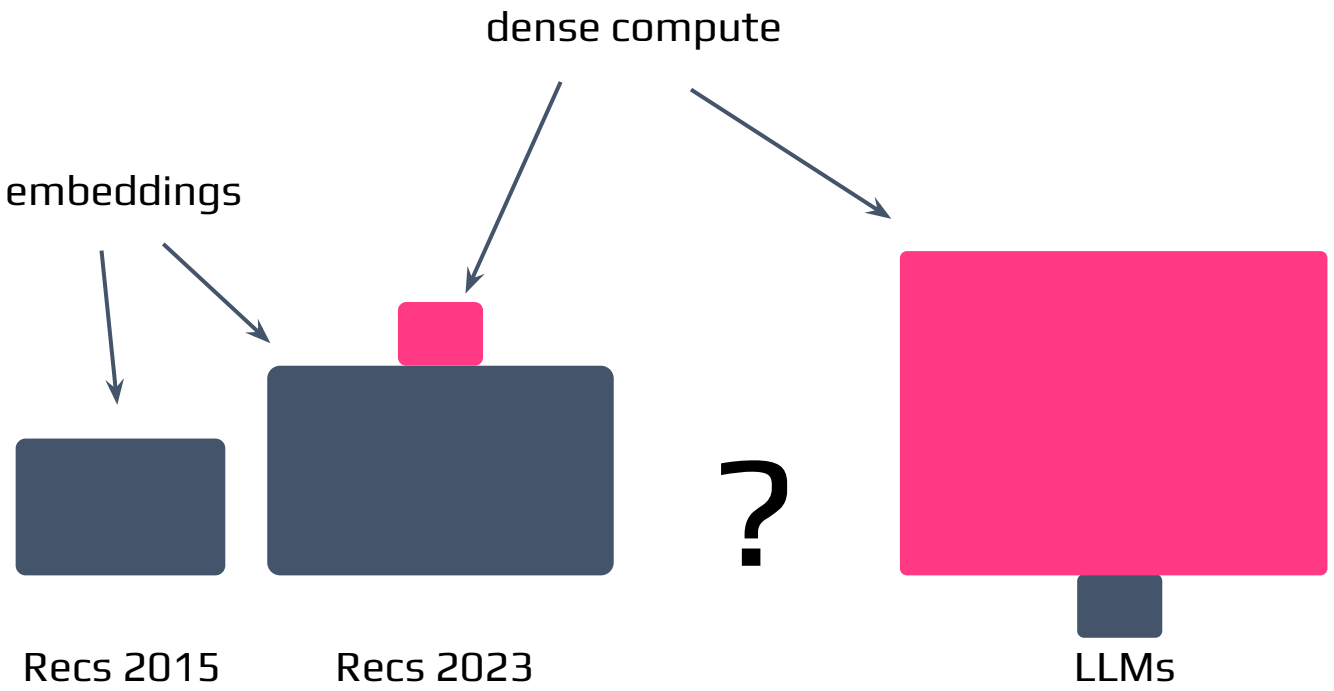


https://arxiv.org/pdf/2104.05158

Содержание

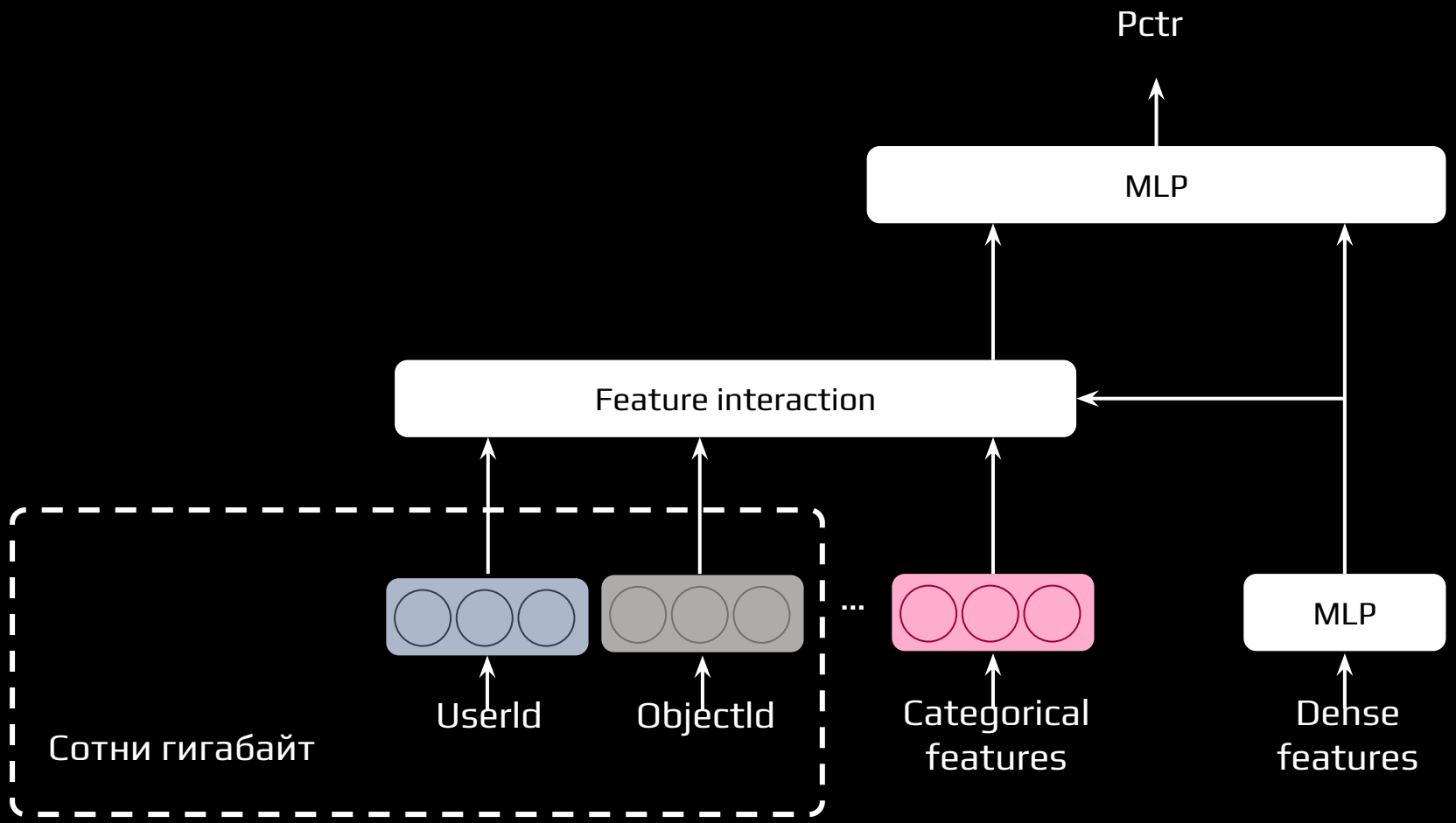
- Проблемы обучения
- Вопросы инференса
  на нагруженном проде
- Результаты внедрения в прод

Большие не только ЛЛМ



https://videorecsys.com/2023/index.html

Пример архитектуры современного нейронного рекома



Масштабы моделей

llama 405
миллиардов
параметров

DCNv2 для рекомендаций
триллионы параметров

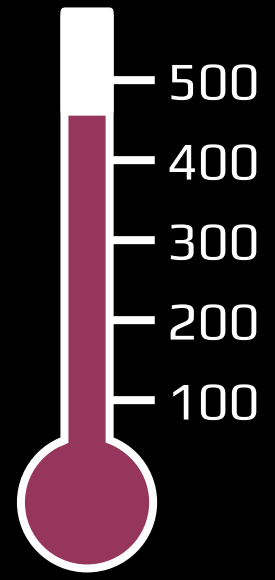
Проблемы обучения

Постановка задачи на искусственном примере

Дано:
DAU = 50 млн
Новых постов ≈ 10 млн/день
Embedding dim = 64
Актуальны объекты созданныеза последний год

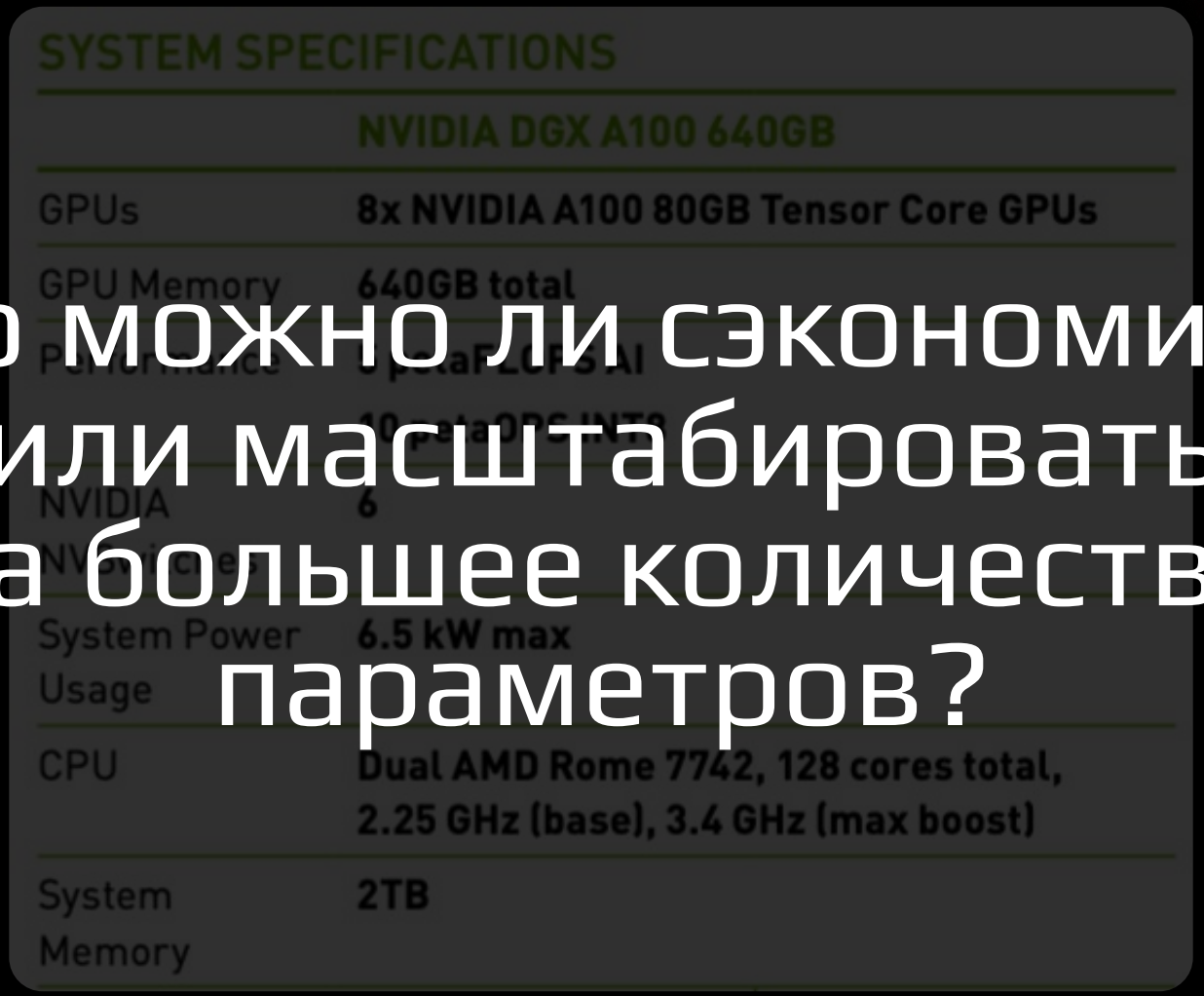
Найти:
Ресурсы=?

Решение:
Объектов = 365 дней * 10 млн/день = 3,65 млрд
Параметров в модели = 3,65 млрд * 64 ≈ 230 млрд
Занимаемая память в fp16 = 230 * 2 = 460 гигабайт



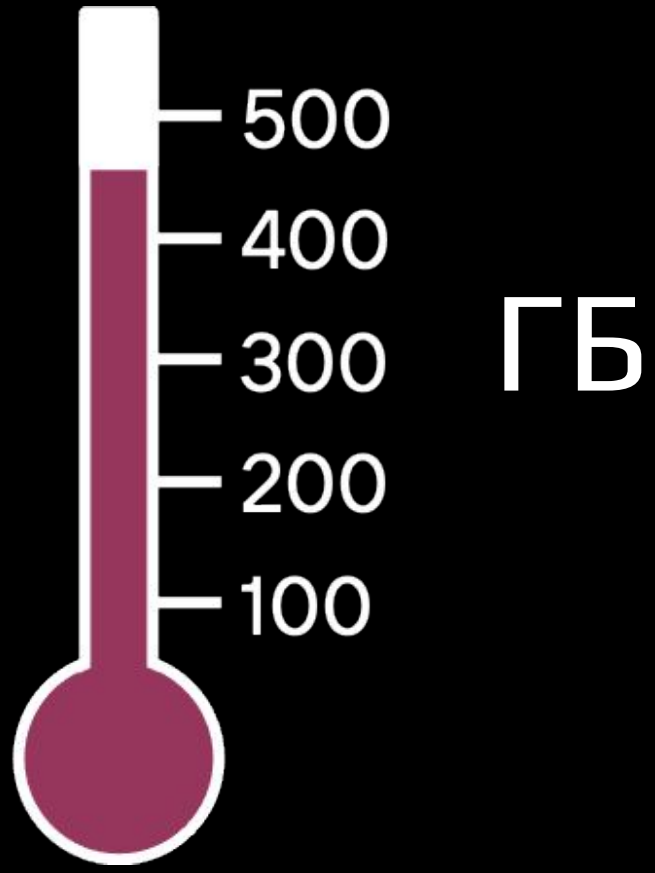
SYSTEM SPECIFICATIONS	
	NVIDIA DGX A100 640GB
GPUs	**8x NVIDIA A100 80GB Tensor Core GPUs**
GPU Memory	**640GB total**
Performance	**5 petaFLOPS AI** **10 petaOPS INT8**
NVIDIA NVSwitches	**6**
System Power Usage	**6.5 kW max**
CPU	**Dual AMD Rome 7742, 128 cores total, 2.25 GHz (base), 3.4 GHz (max boost)**
System Memory	**2TB**

Существует железо способное решить эту задачу «в лоб»



Но можно ли сэкономить или масштабировать на большее количество параметров?

Затраты по памяти

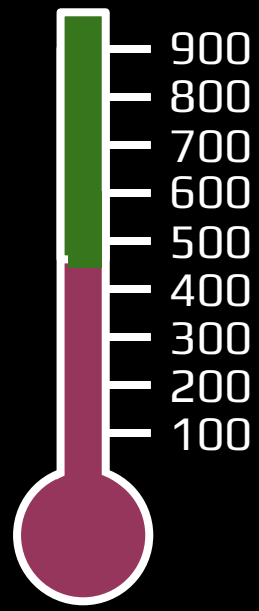


RowWise оптимизатор

Сокращение расходов на хранение стейта оптимизатора во время обучения:

460 на эмбеддинги + 460 на стейт оптимизатора = 920 ГБ
460 эмбеддингов + 460/64 оптимизатор ≈ 467 ГБ

->



	m_t	V_t	Sparse Friendly	Memory Requirement
SGD	g_t	$\mathbf{1}^2$	N	0
AdaGrad [31]	g_t	$\sum_{\tau=1}^{t} g_\tau^2$	Y	d
Adam [32]	$\beta_1 m_{t-1} + (1-\beta_1)\, g_t$	$\beta_2 V_{t-1} + (1-\beta_2)\, g_t^2$	Y	$2d+1$
rAdaGrad	g_t	$\sum_{\tau=1}^{t} \|\|g_t\|\|_2^2/d * \mathbf{1}$	Y	1

https://storage.cs.tsinghua.edu.cn/papers/sc20-kraken.pdf

RowWise оптимизатор реализация

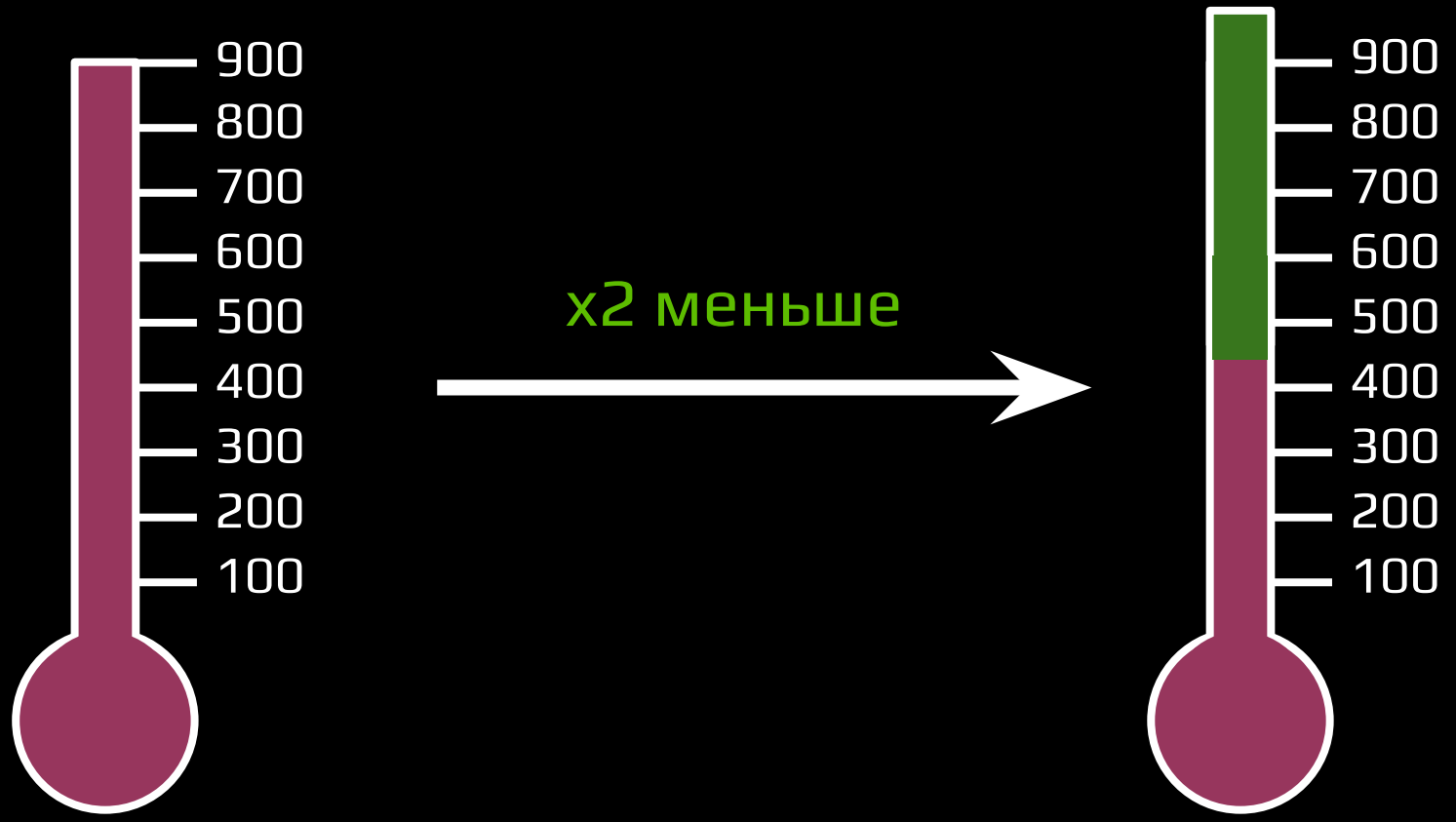
Dense Opt	Sparse Opt	Memory Usage	Criteo			
			DNN	W&D	DeepFM	DCN
SGD		1x	0.7979	0.7896	0.7986	0.7908
AdaGrad		2x	0.8001	0.7899	0.8016	0.7992
Adam		3x	0.8066	0.7893	0.7956	0.7955
Adam	AdaGrad	~2x	0.8048	0.8005	0.8057	0.8044
Adam	SGD	~1x	0.7974	**0.7988**	0.8038	0.8026
Adam	rAdaGrad	~1x	**0.8010**	0.7907	**0.8048**	**0.8048**

AUC

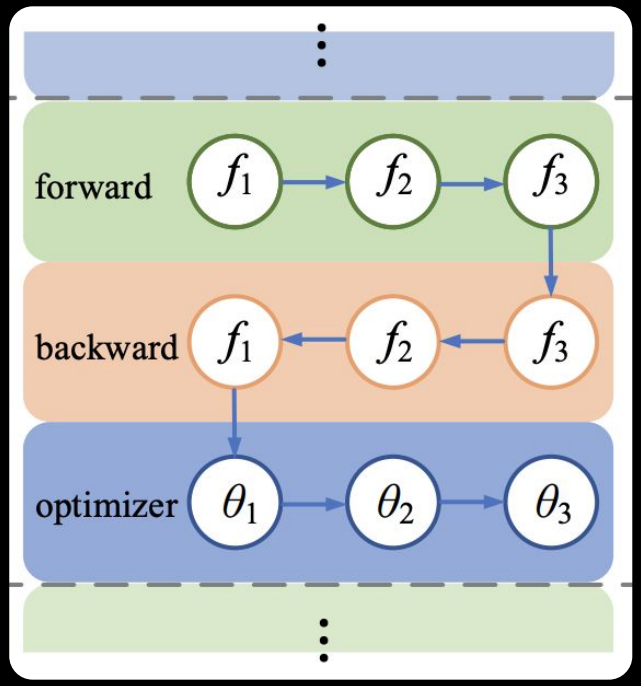
```
# update step
step_t += 1
step = step_t.item()
grad = grad if not maximize else -grad

row_wise_grad = grad.mean(axis=1).view(-1, 1)
if weight_decay != 0:
    grad = grad.add(param, alpha=weight_decay)
    row_wise_grad = grad.add(param, alpha=weight_decay)

clr = lr / (1 + (step - 1) * lr_decay)

state_sum.addcmul_(row_wise_grad, row_wise_grad, value=1)
std = state_sum.sqrt().add_(eps)
param.addcdiv_(row_wise_grad, std, value=-clr)
```

https://github.com/pytorch/torchrec/blob/main/torchrec/optim/rowwise_adagrad.py

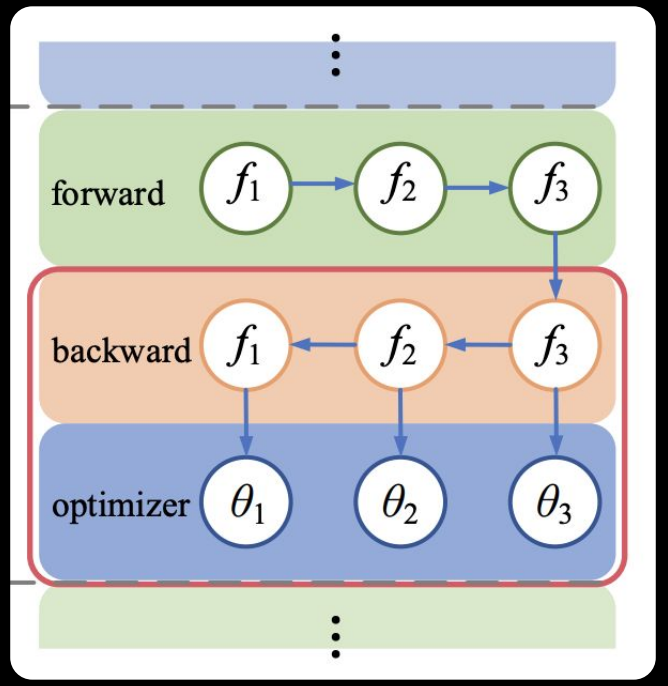
Затраты по памяти
с RowWise оптимизатором



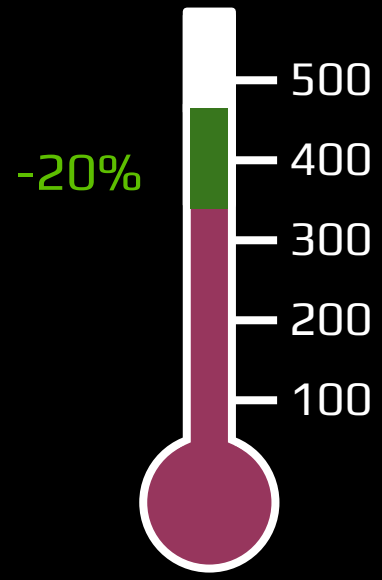
Backward-fusion



Baseline



Backward-fusion



https://www.jqgu.net/publications/papers/ML_ICLRW2021_Gu.pdf

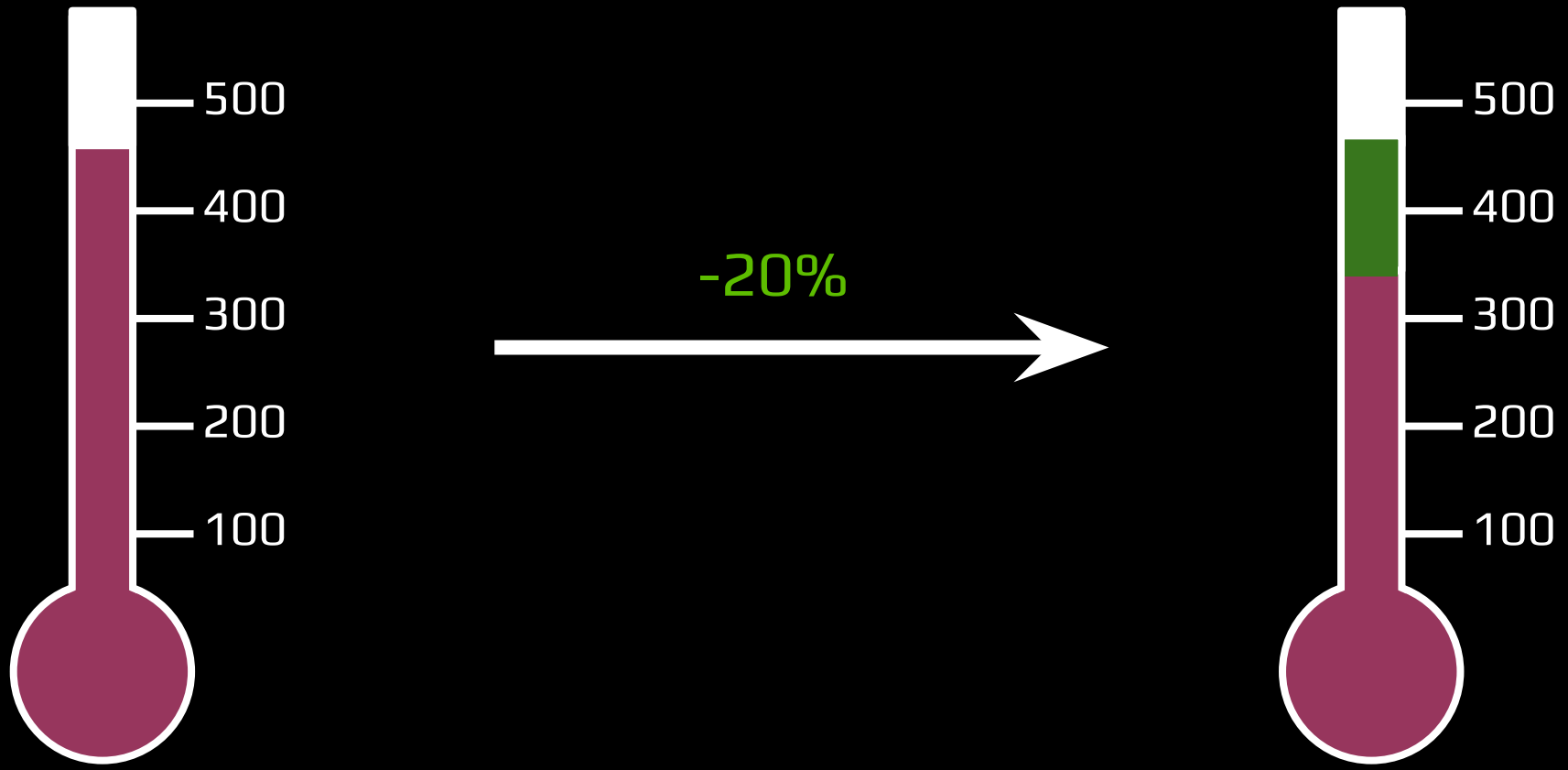
Backward-fusion реализация

```
# Вызов forward и backward
loss = model.forward(image)
loss.sum().backward()

# Шаг оптимизатора
optimizer.step()
optimizer.zero_grad()
```

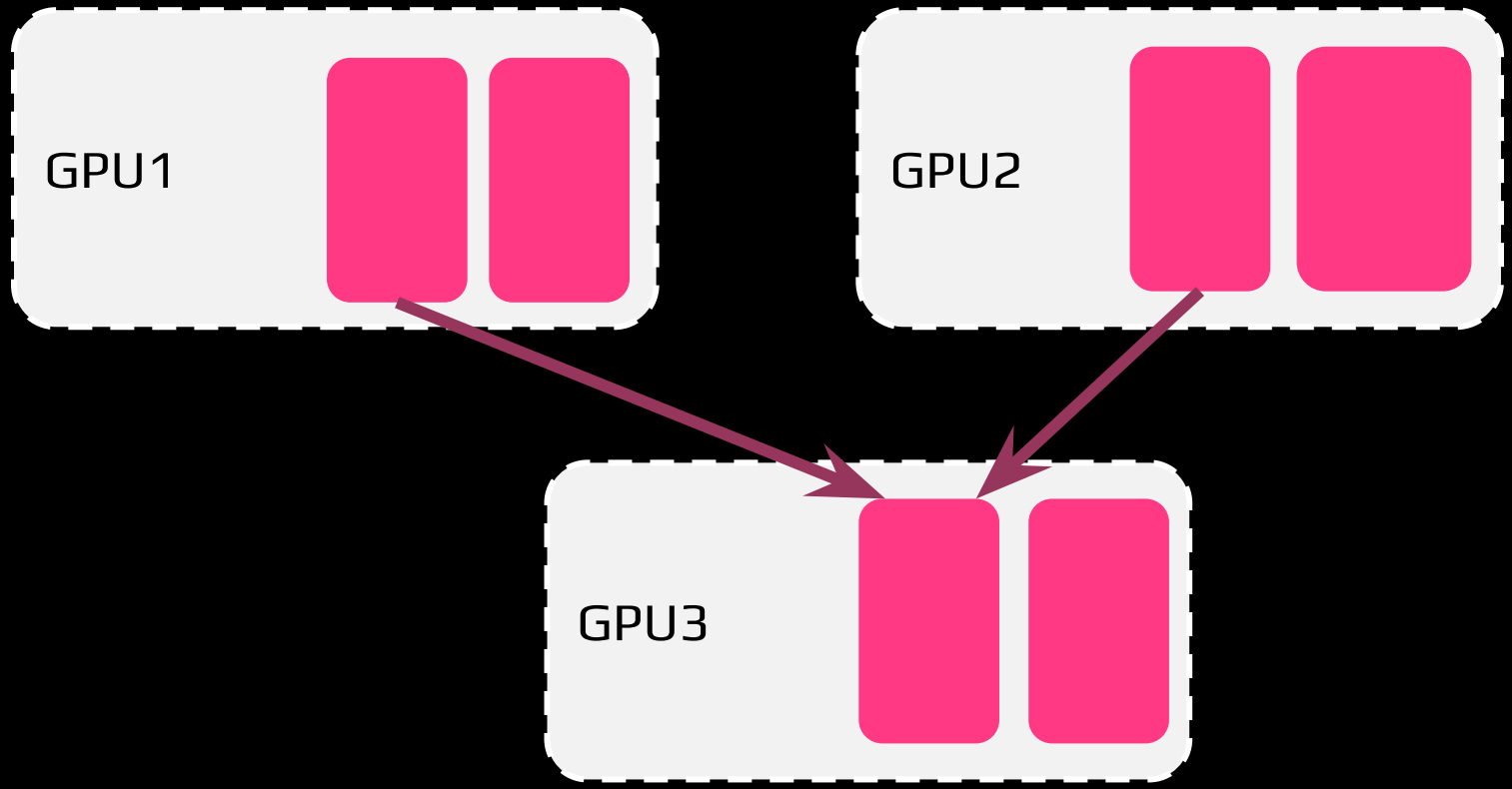
```
# Словарь оптимизаторов для всех параметров
optimizer_dict = {p: torch.optim.Adam([p], foreach=False)
for p in model.parameters()}

# Хук для вызова ``step()`` и ``zero_grad()``
def optimizer_hook(parameter) -> None:
  optimizer_dict[parameter].step()
  optimizer_dict[parameter].zero_grad()

# Регистрируем хук для всех параметров
for p in model.parameters():
   p.register_post_accumulate_grad_hook(optimizer_hook)

# Вызов forward и backward
loss = model.forward(image)
loss.sum().backward()
```

https://pytorch.org/tutorials/intermediate/optimizer_step_in_backward_tutorial.html

Затраты по памяти с Backward-fusion



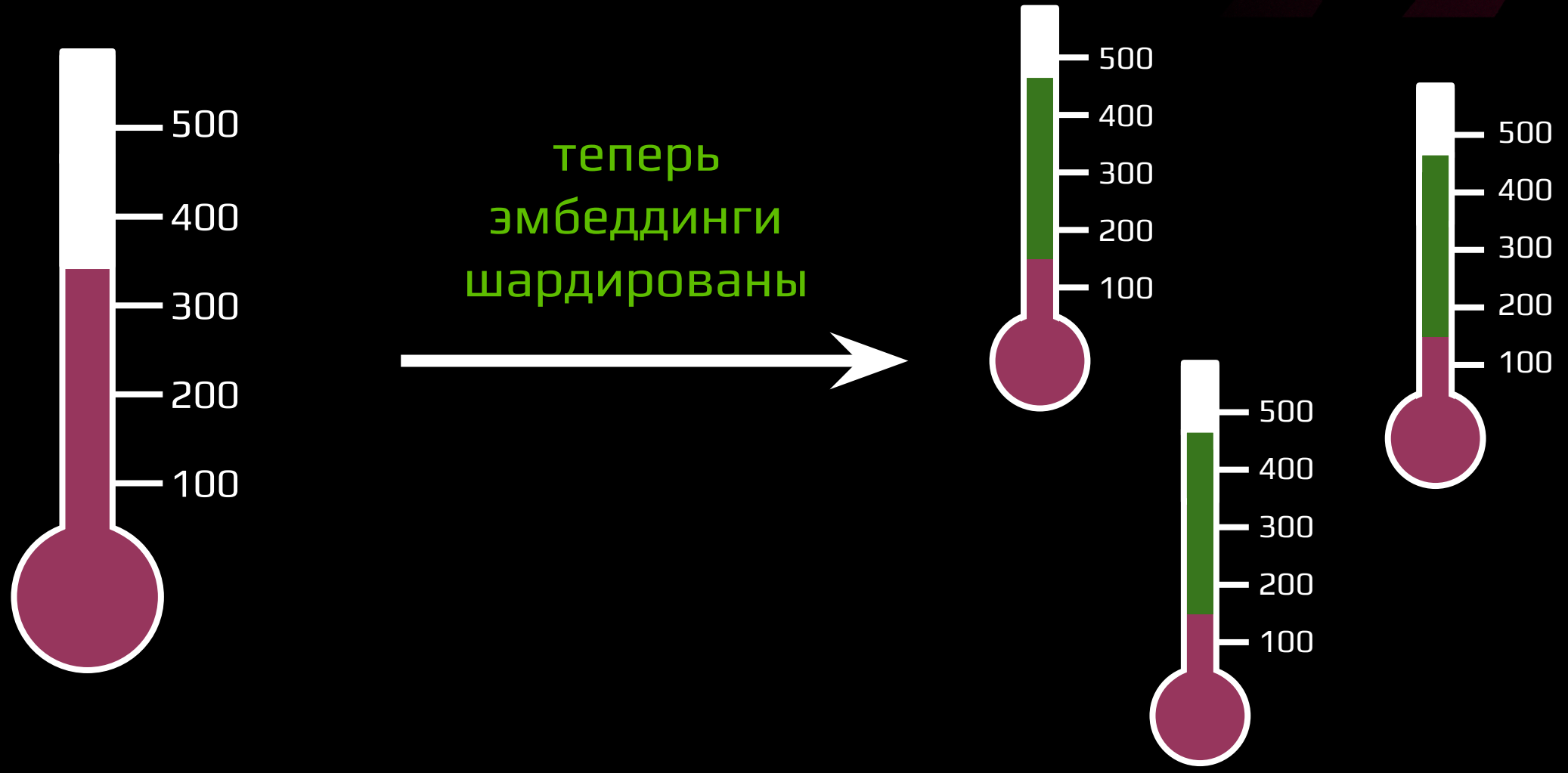
Model parallelism



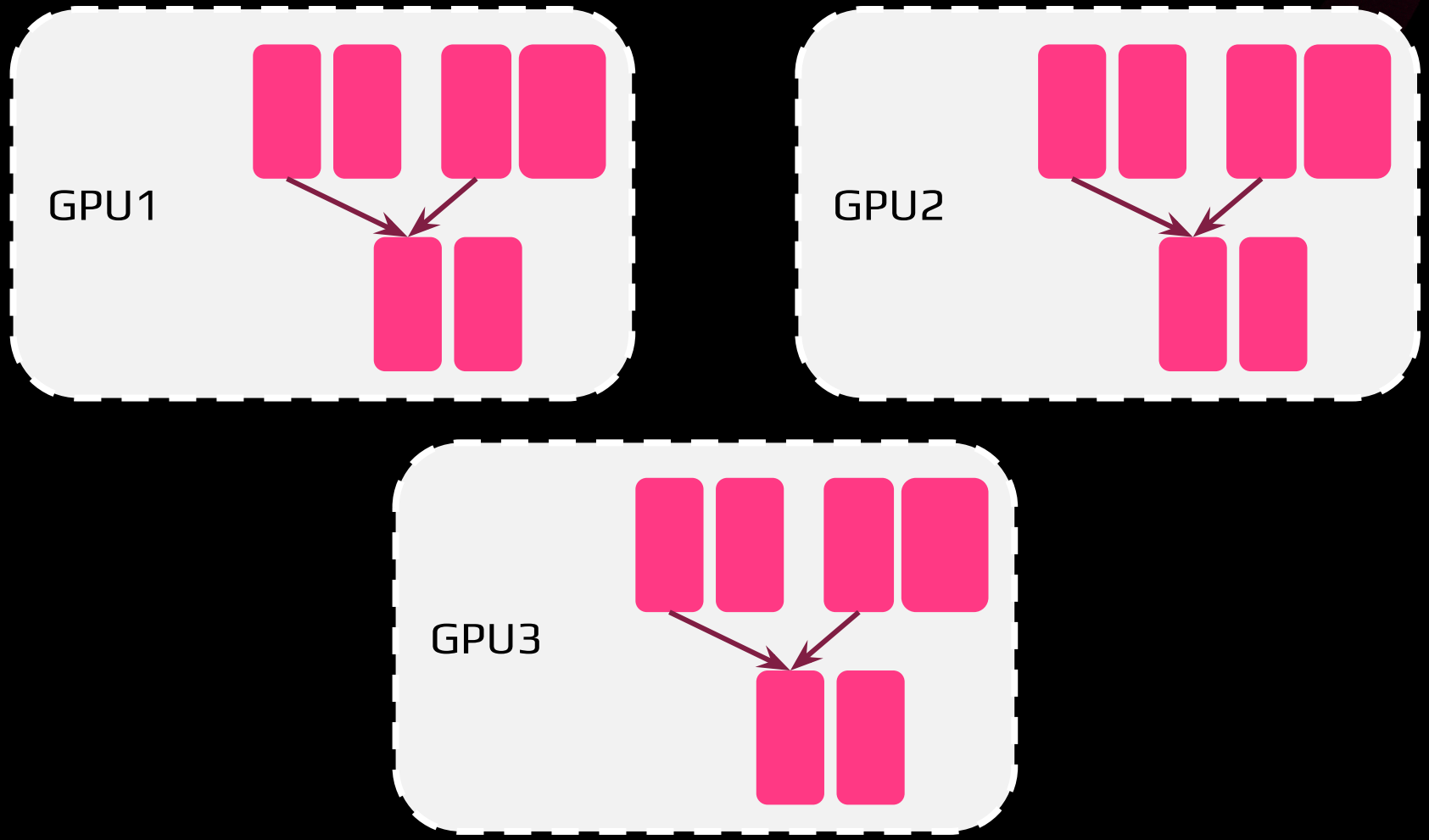
Model parallelism реализация на 1 ноде

```
class ModelParallel(nn.Module):
    def __init__(self):
        super().__init__()

        self.seq1 = nn.Sequential(
            self.layer1,
            self.layer2
        ).to('cuda:0')

        self.seq2 = nn.Sequential(
            self.layer3,
            self.layer4,
        ).to('cuda:1')

        self.fc.to('cuda:1')

    def forward(self, x):
        x = self.seq2(self.seq1(x).to('cuda:1'))
        return self.fc(x.view(x.size(0), -1))
```

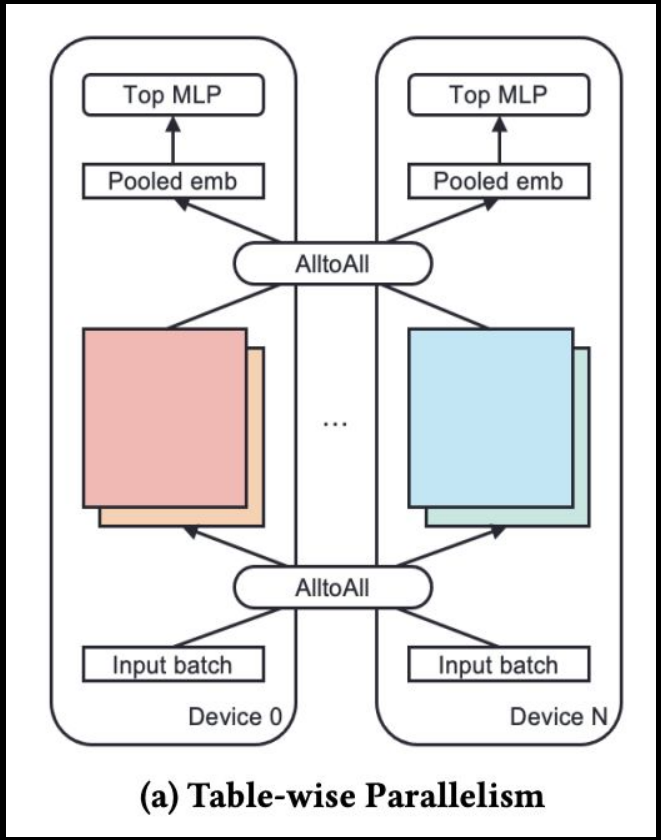
Затраты по памяти Model parallelism



Data parallelism



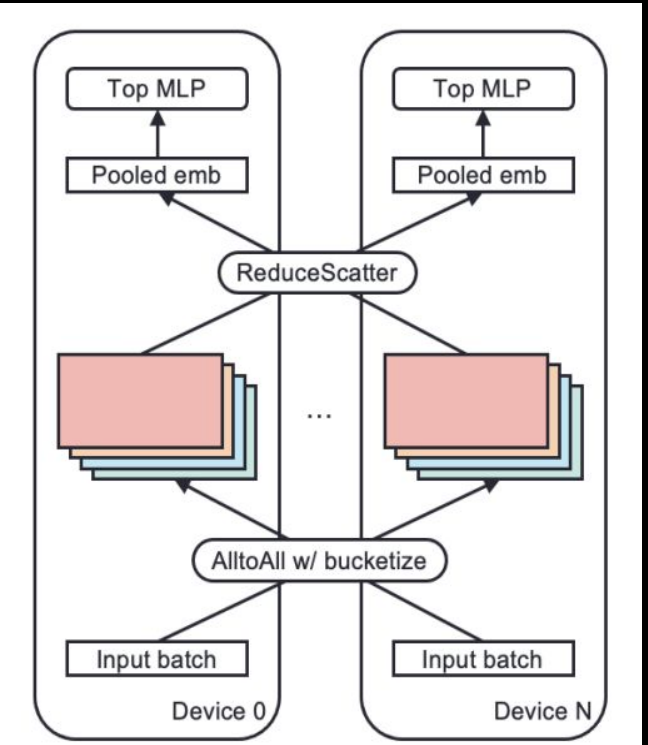
Parallelism table-wise



(a) Table-wise Parallelism

https://arxiv.org/pdf/2104.05158

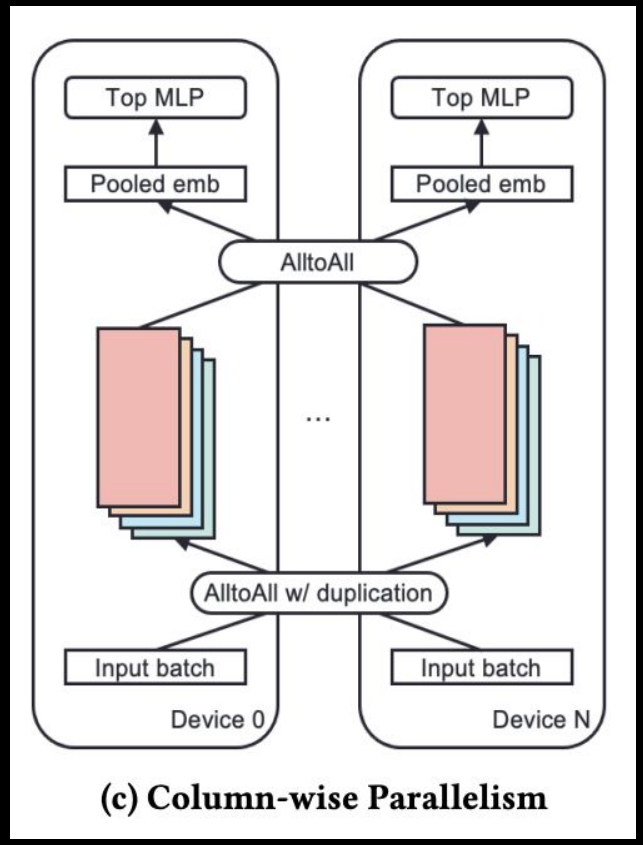
Parallelism row-wise



(b) Row-wise Parallelism

https://arxiv.org/pdf/2104.05158

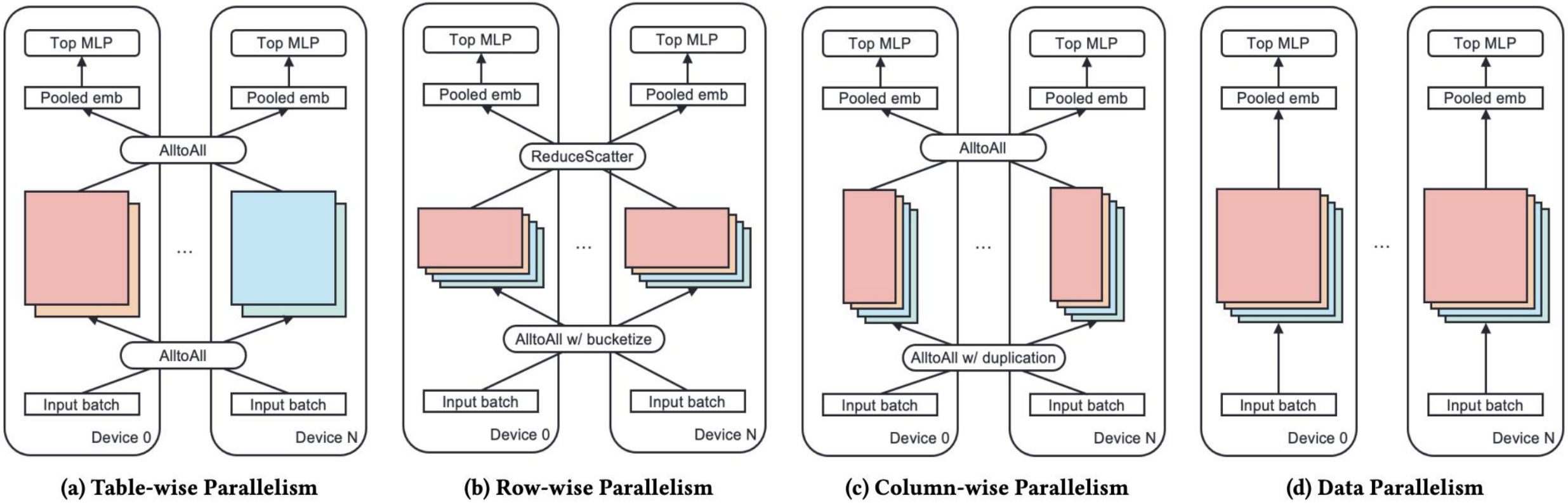
Parallelism column-wise



(c) Column-wise Parallelism

https://arxiv.org/pdf/2104.05158

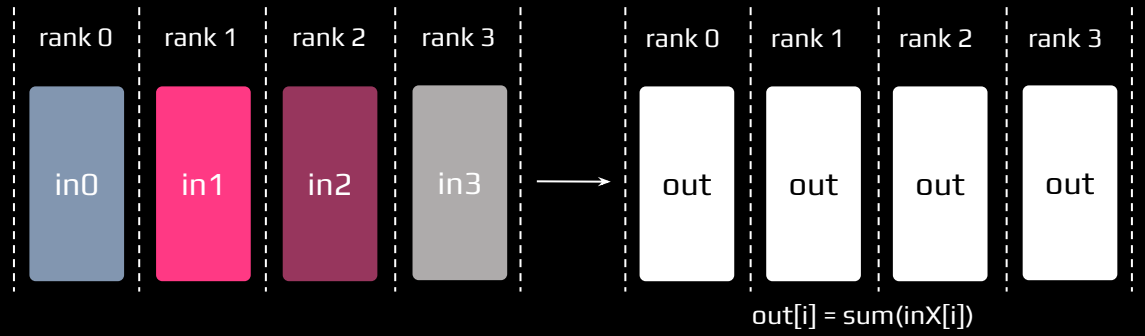
Parallelism виды



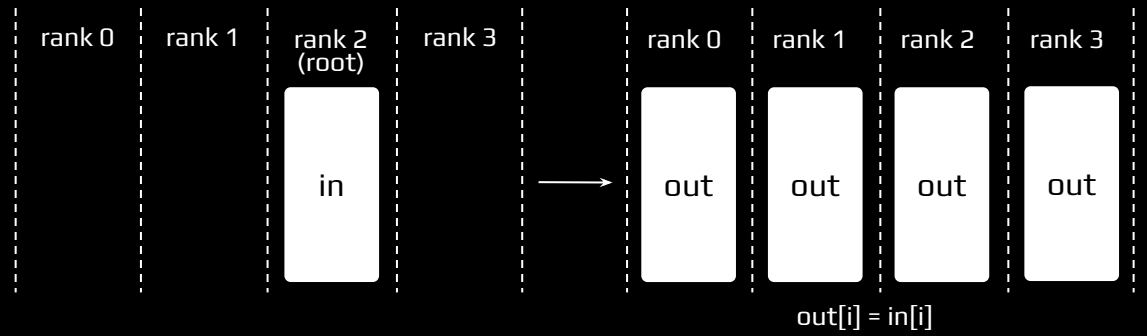
https://arxiv.org/pdf/2104.05158

Parallelism реализация для мульти нод сценария NCCL

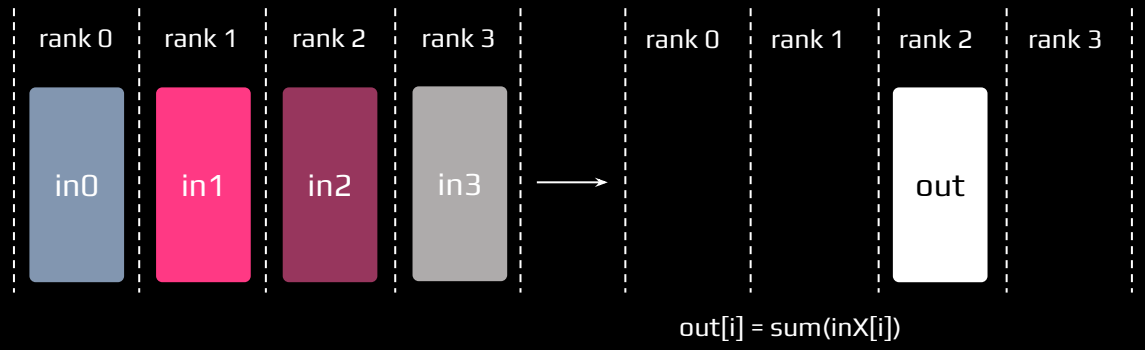
AllReduce



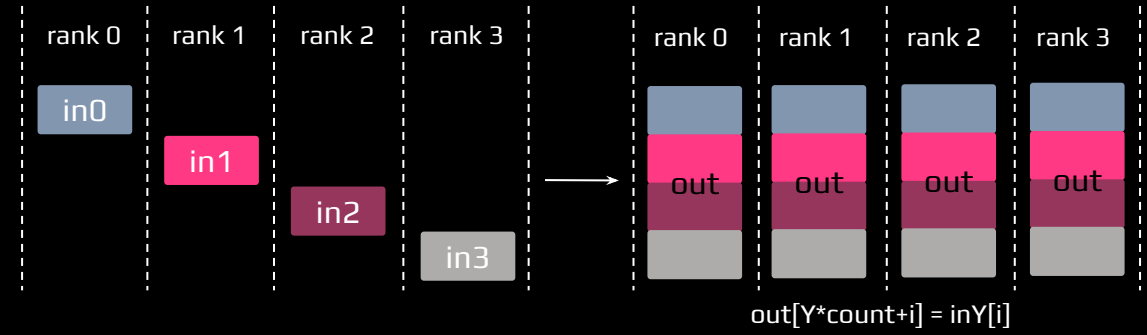
Broadcast



Reduce



AllGather



https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/overview.html

Parallelism реализация для мульти нод сценария

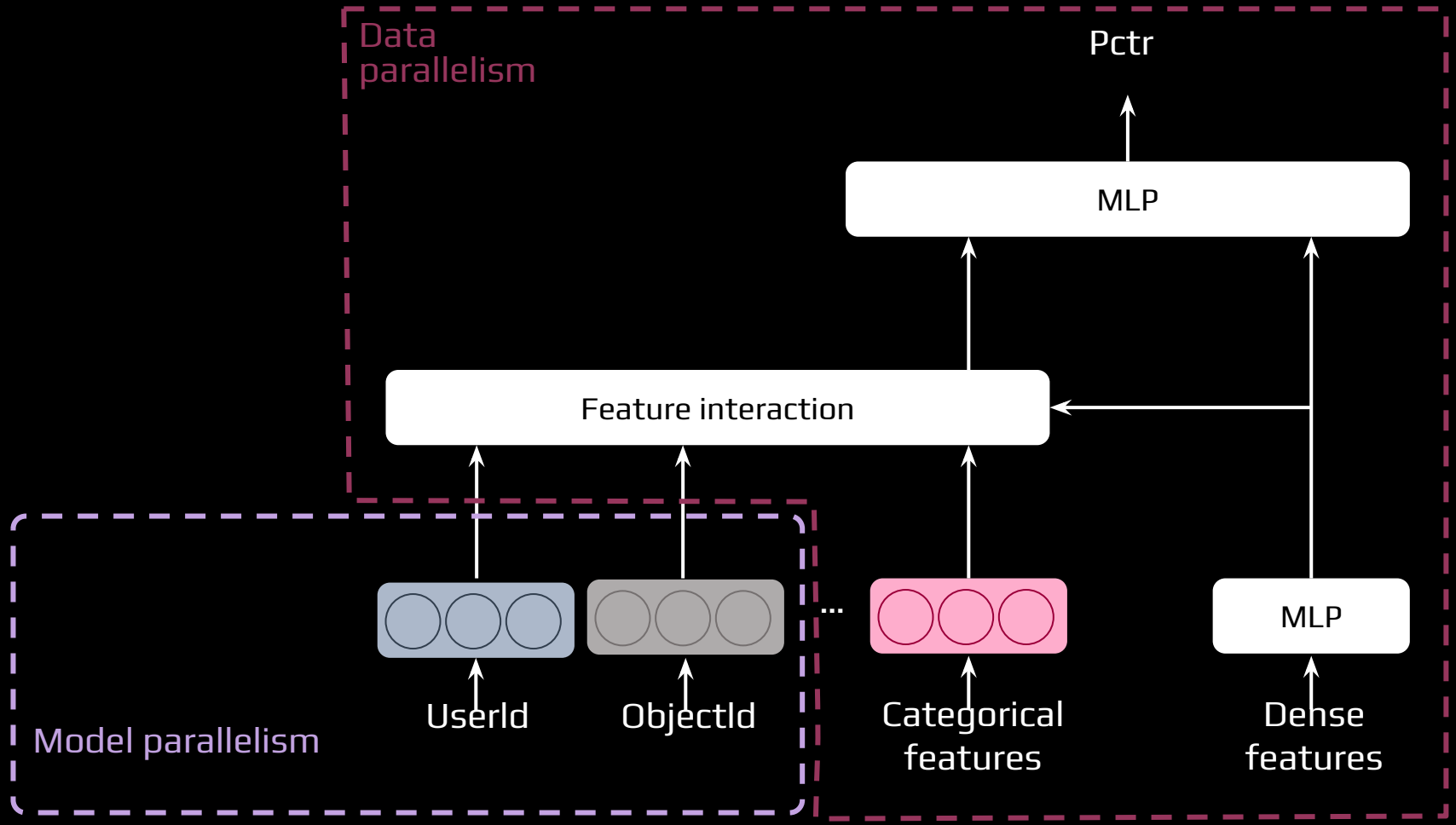
torchrec / torchrec / distributed / sharding / rw_sequence_sharding.py

Code | Blame | 326 lines (291 loc) · 11.2 KB

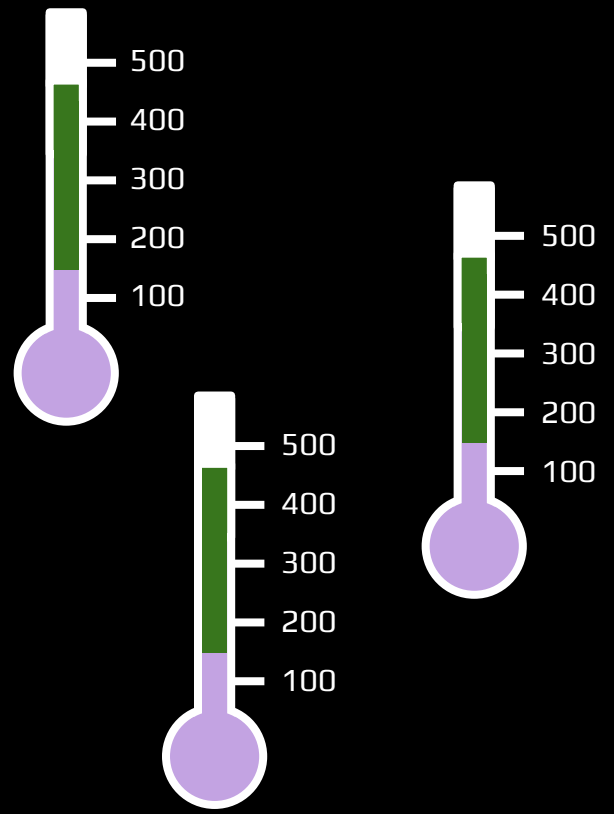
```python
class RwSequenceEmbeddingDist(
    def __init__(
        self,
        pg: dist.ProcessGroup,
        num_features: int,
        device: Optional[torch.device] = None,
        qcomm_codecs_registry: Optional[Dict[str, QuantizedCommCodecs]] = None,
    ) -> None:
        super().__init__()
        self._dist = SequenceEmbeddingsAllToAll(
            pg,
            [num_features] * pg.size(),
            device,
            codecs=(
                qcomm_codecs_registry.get(
                    CommOp.SEQUENCE_EMBEDDINGS_ALL_TO_ALL.name, None
                )
                if qcomm_codecs_registry
                else None
            ),
        )
```

https://github.com/pytorch/torchrec/blob/main/torchrec/distributed/sharding/rw_sequence_sharding.py

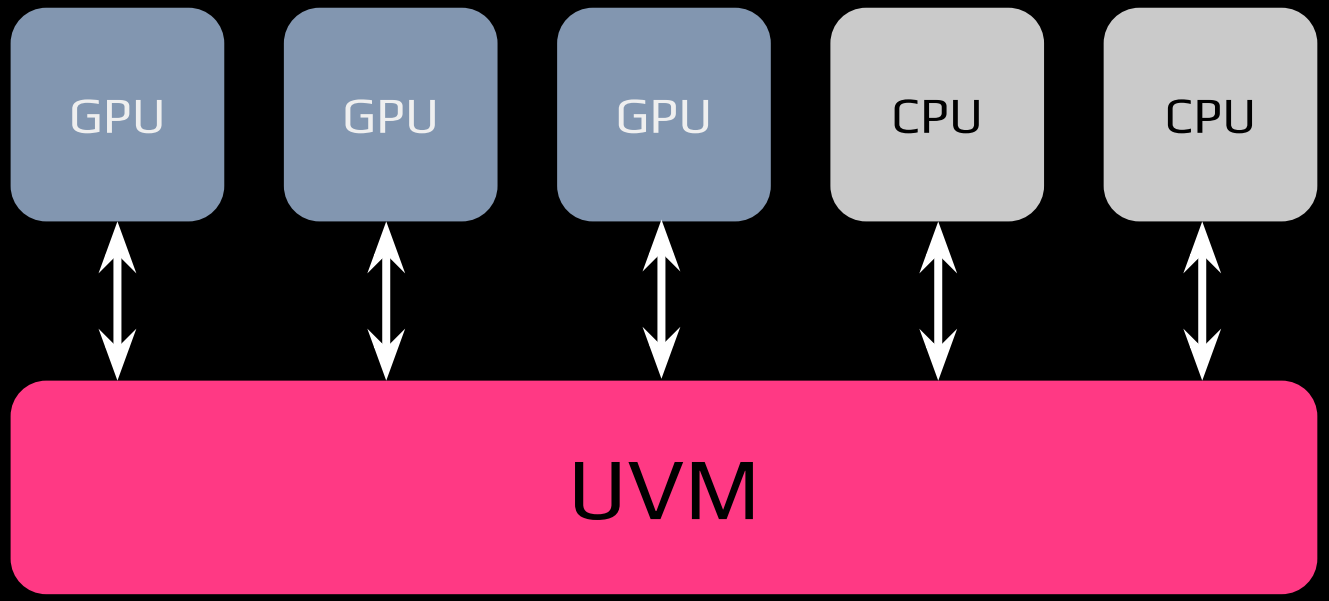
Model/Data parallelism

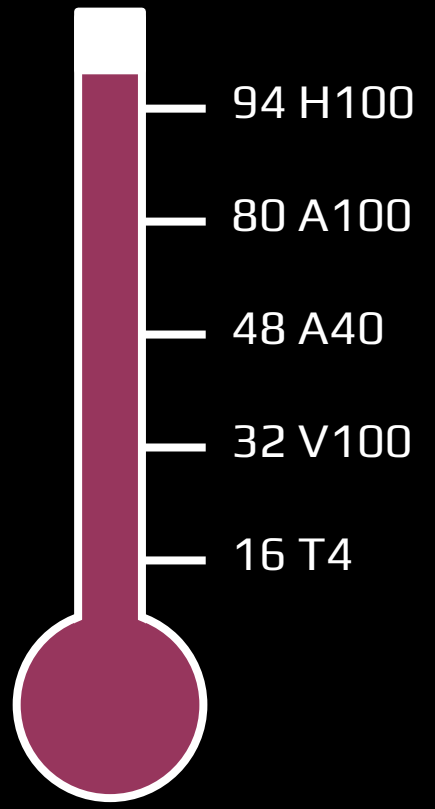


Теперь шардированно по разным GPU



Unified Virtual Memory





Использование Unified Virtual Memory

__host__ cudaError_t cudaMallocManaged (void** devPtr , size_t size , unsigned int flags = cudaMemAttachGlobal)

Allocates memory that will be automatically managed by the Unified Memory system.

FBGEMM / fbgemm_gpu / src / memory_utils / **memory_utils.cu**

Code | Blame | 453 lines (378 loc) · 14.4 KB

```
86    // Allocate the ATen Tensor with unified managed memory (UVM)
87    Tensor new_managed_tensor_internal(
88        const Tensor& self,
89        const std::vector<std::int64_t>& sizes) {
90      CUDA_DEVICE_GUARD(self);
91
92      auto strides = defaultStrides(sizes);
93      size_t size_bytes =
94          at::detail::computeStorageNbytes(sizes, strides, self.dtype().itemsize());
95      void* ptr;
96      AT_CUDA_CHECK(cudaMallocManaged(&ptr, size_bytes));
```

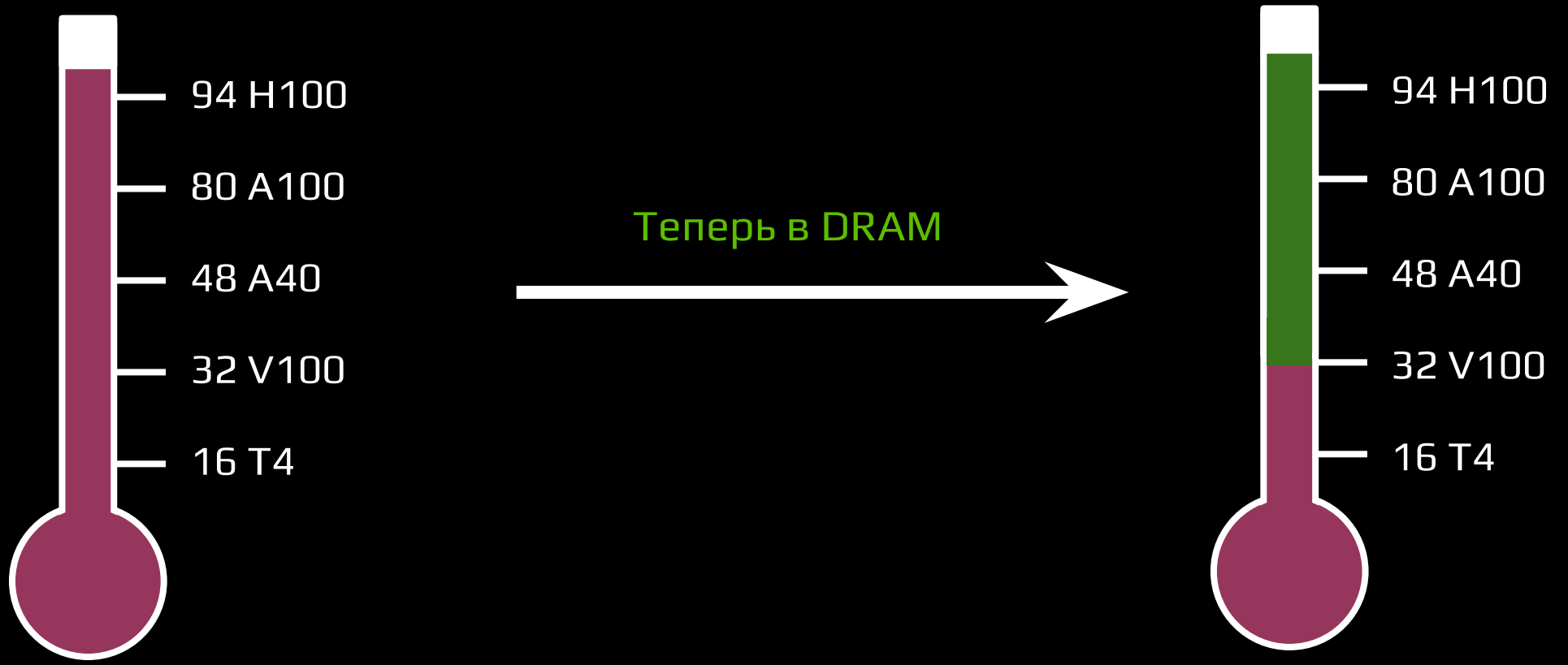
https://github.com/pytorch/FBGEMM/blob/main/fbgemm_gpu/src/memory_utils/memory_utils.cu

Использование Unified Virtual Memory

```
from torchrec.distributed.planner import EmbeddingShardingPlanner, Topology
from torchrec.distributed.planner.types import ParameterConstraints
from torchrec.distributed.embedding_types import EmbeddingComputeKernel
from torchrec.distributed.embeddingbag import EmbeddingBagCollectionSharder
from torchrec.distributed.types import ModuleSharder


topology = Topology(world_size=1, compute_device="cuda")
constraints = {
    "large_table": ParameterConstraints(
        sharding_types=["table_wise"],
        compute_kernels=[EmbeddingComputeKernel.BATCHED_FUSED_UVM.value],
    )
}
plan = EmbeddingShardingPlanner(
    topology=topology, constraints=constraints
).plan(
    ebc, [cast(ModuleSharder[torch.nn.Module], EmbeddingBagCollectionSharder())]
)

uvm_model = torchrec.distributed.DistributedModelParallel(
    ebc,
    device=torch.device("cuda"),
    plan=plan
)
```

https://github.com/pytorch/torchrec/blob/main/examples/sharding/uvm.ipynb

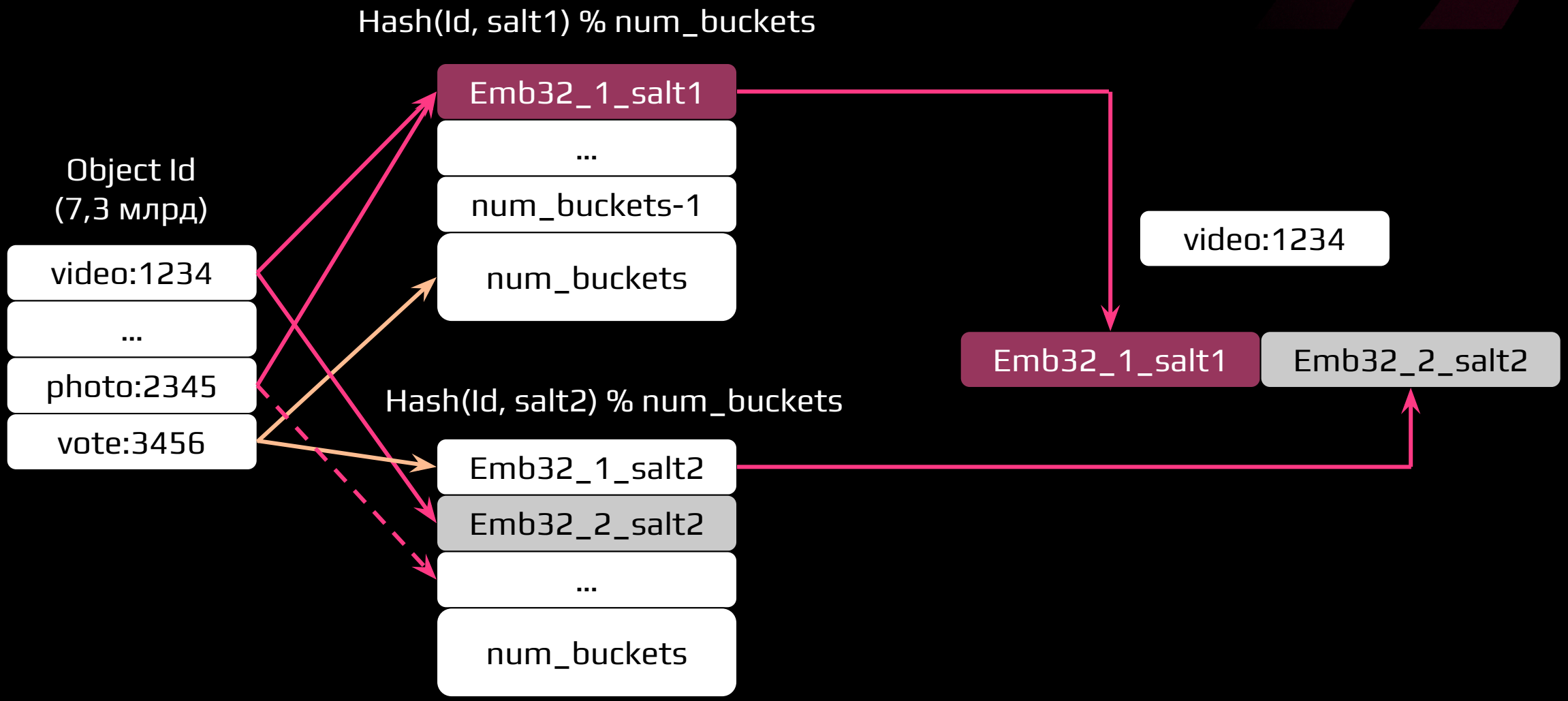
Затраты по памяти и UVM



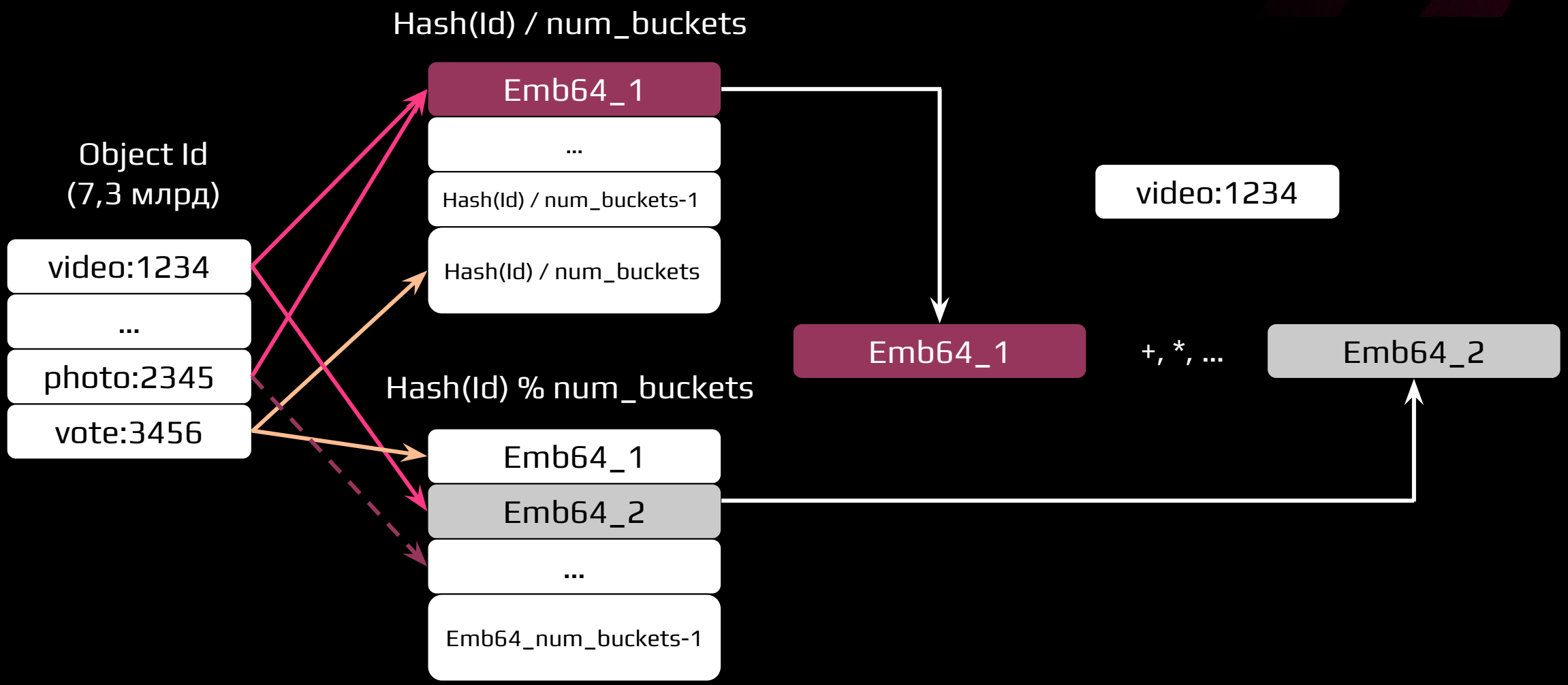
Что если данных резко станет больше?

Что делать, если данных станет больше?

Hashing Trick Product Quantization



Hashing Trick Quotient Remainder



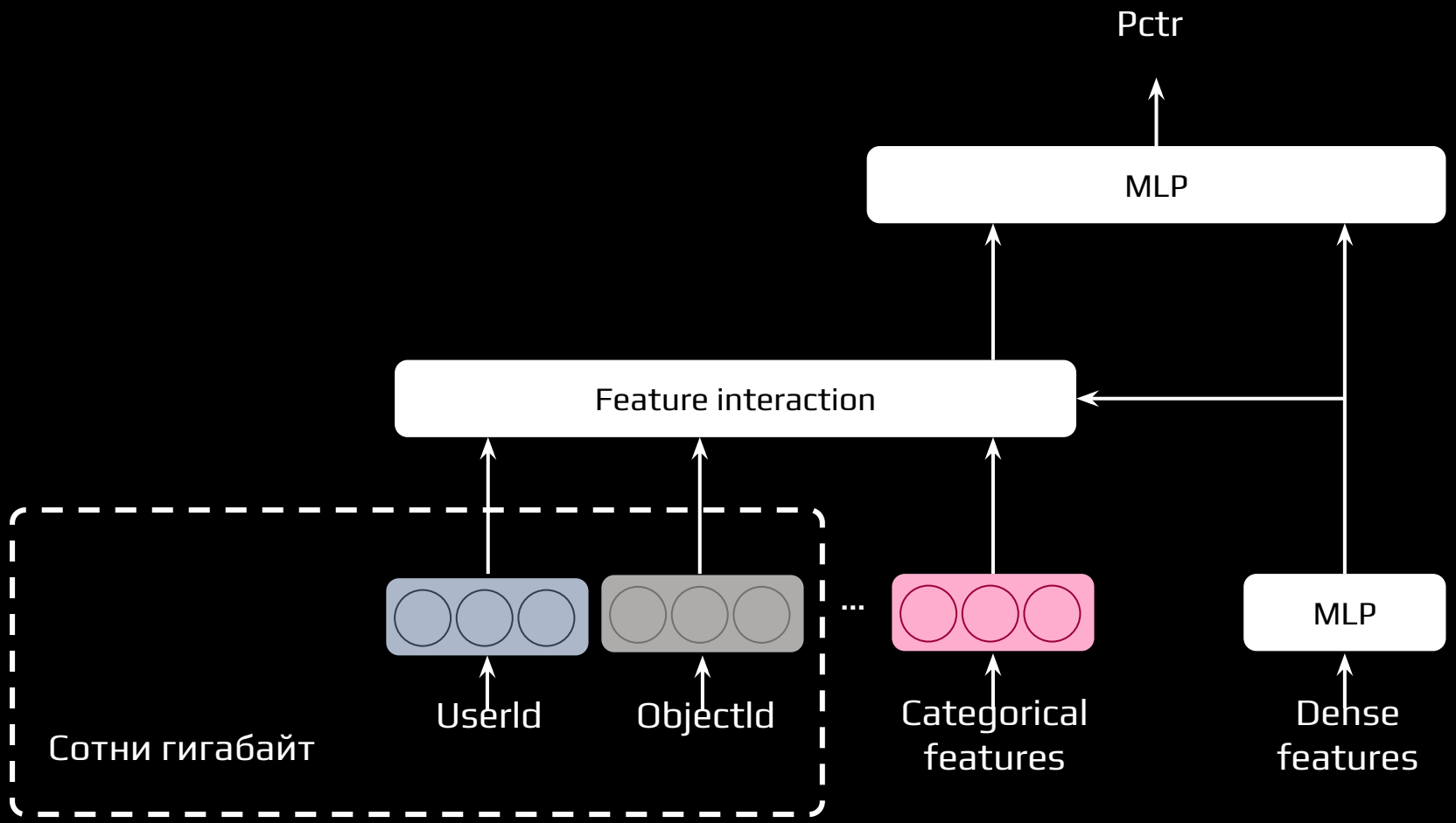
Проблемы инференса на нагруженном проде

Задача: ранжирование подписной ленты соцсети

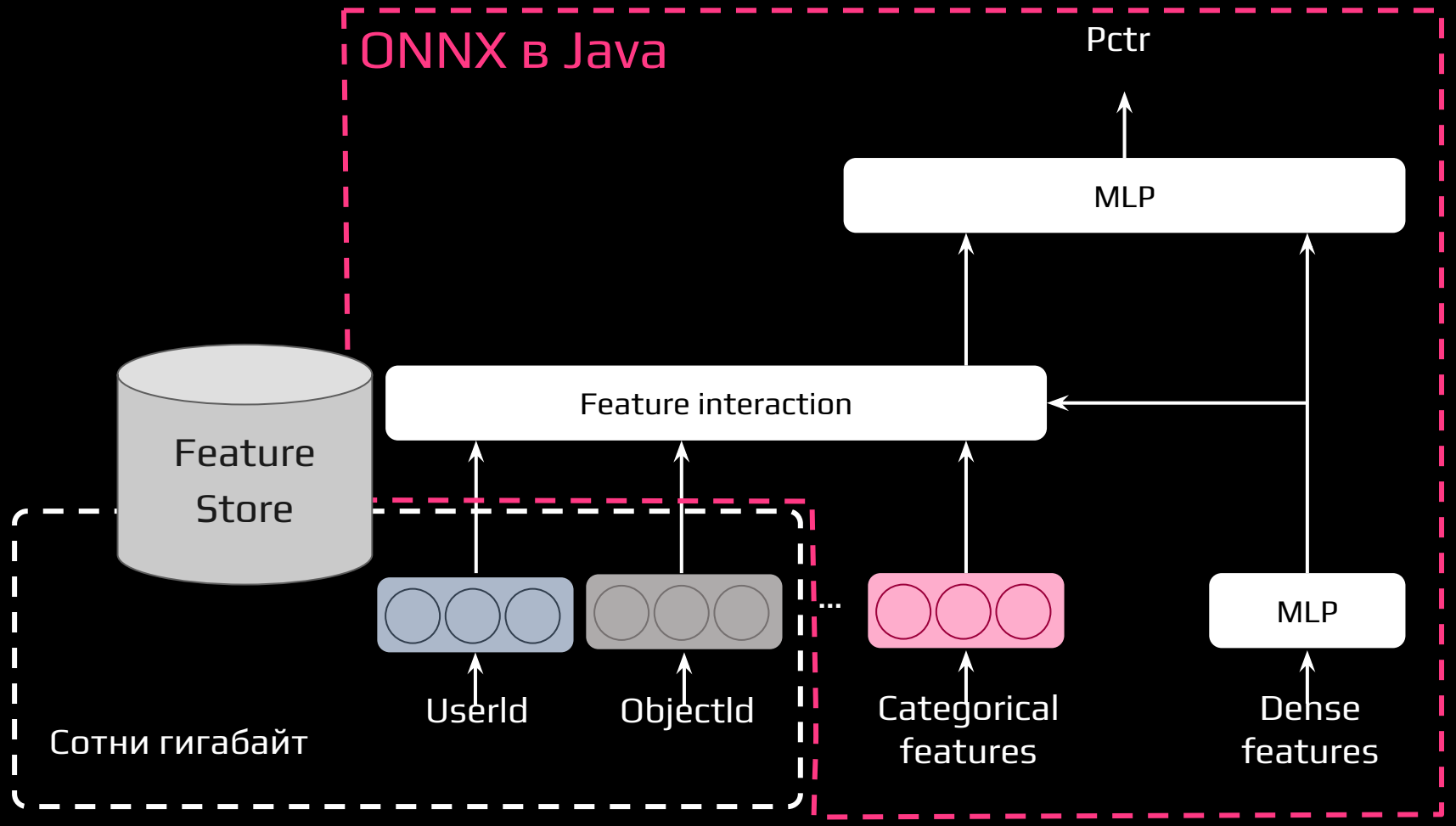
Типы контента (фидов) в ленте:

- Контент друзей (новые посты, фото, видео, комментарии, репосты и тд)
- Контент групп (посты тематических групп, новости, развлекательный контент, локальные новости и тд)
- Второй круг (посты вида “Ваш друг считает классным”)

Инференс модели в проде



Инференс модели в проде ОК 10к rps



https://www.youtube.com/watch?v=UHI7JHUHD0s

odnoklassniki-ml

Реализует и поддерживает:

- схему данных
- работу с датасетам
- поддержку векторов и тензоров
- пайплайны для трансформации датасета
- скриптовый язык для реализации кастомных трансформаций
- движок выполнения пайплайнов с поддержкой асинхронных запросов
- сериализацию и выполнение пайплайнов для Spark
- декларативное описания фич и автоматическое построение пайплайна для их вычисления

https://github.com/odnoklassniki
https://t.me/mlvok

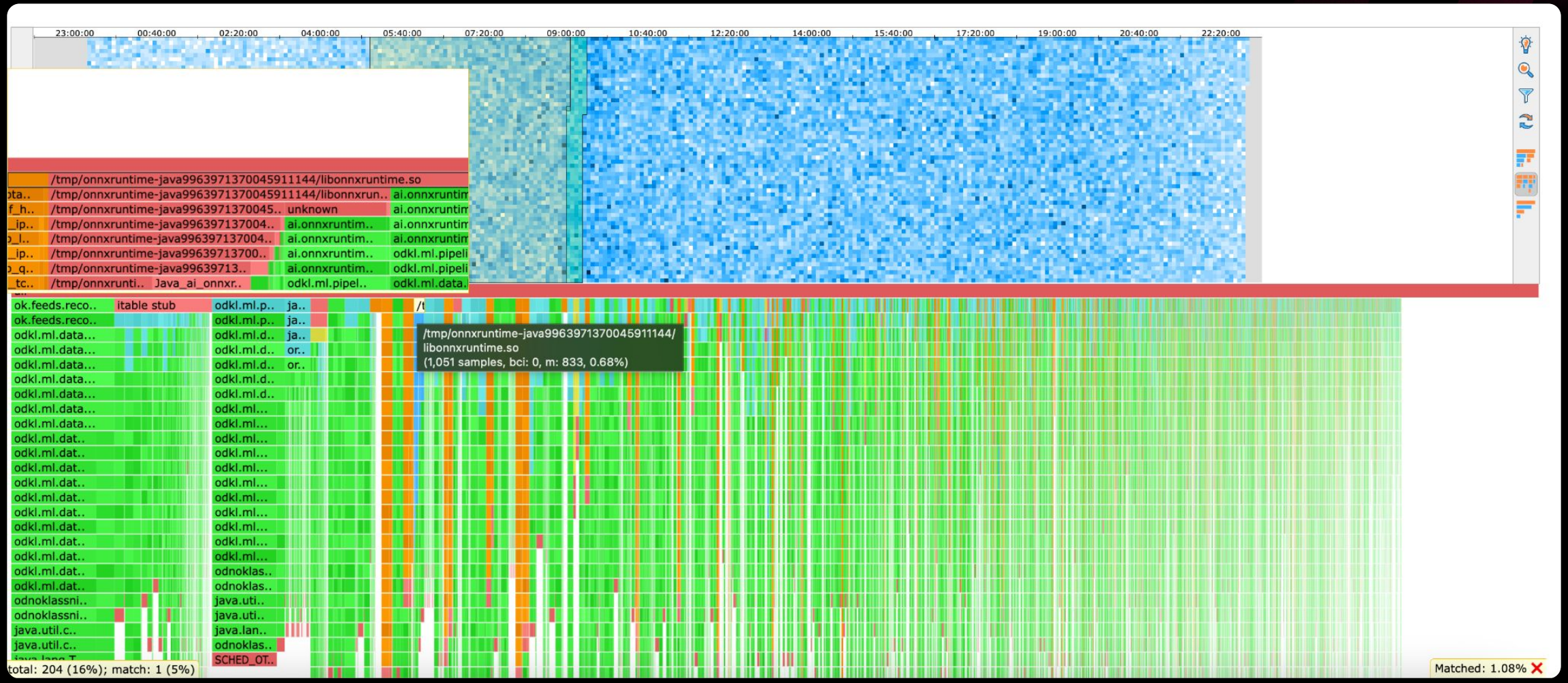
Существующие модели

Для сравнения ResNet-18 для обработки изображения 224 x 224 потребует 233 GFLOPs

Baseline	Criteo					
	Logloss	AUC	Params	FLOPS	Best Setting	
PNN	0.4421 (5.8E-4)	0.8099 (6.1E-4)	3.1M	6.1M	(3, 1024)	OPNN
DeepFm	0.4420 (1.4E-4)	0.8099 (1.5E-4)	1.4M	2.8M	(2, 768)	–
DLRM	0.4427 (3.1E-4)	0.8092 (3.1E-4)	1.1M	2.2M	(2, 768)	[512,256,64]
xDeepFm	0.4421 (1.6E-4)	0.8099 (1.8E-4)	3.7M	32M	(3, 1024)	l=2, n=100
AutoInt+	0.4420 (5.7E-5)	0.8101 (2.6E-5)	4.2M	8.7M	(4, 1024)	l=2, h=2, e=40
DCN	0.4420 (1.6E-4)	0.8099 (1.7E-4)	2.1M	4.2M	(2, 1024)	l=4
DNN	0.4421 (6.5E-5)	0.8098 (5.9E-5)	3.2M	6.3M	(3, 1024)	–
Ours						
DCN-V2	**0.4406 (6.2E-5)**	**0.8115 (7.1E-5)**	3.5M	7.0M	(2, 768)	l=2
DCN-Mix	0.4408 (1.0E-4)	0.8112 (9.8E-5)	2.4M	4.8M	(2, 512)	l=3, K=4, r=258
CrossNet	0.4413 (2.5E-4)	0.8107 (2.4E-4)	2.1M	4.2M	–	l=4, K=4, r=258

https://arxiv.org/abs/2008.13535

Профилирование Java



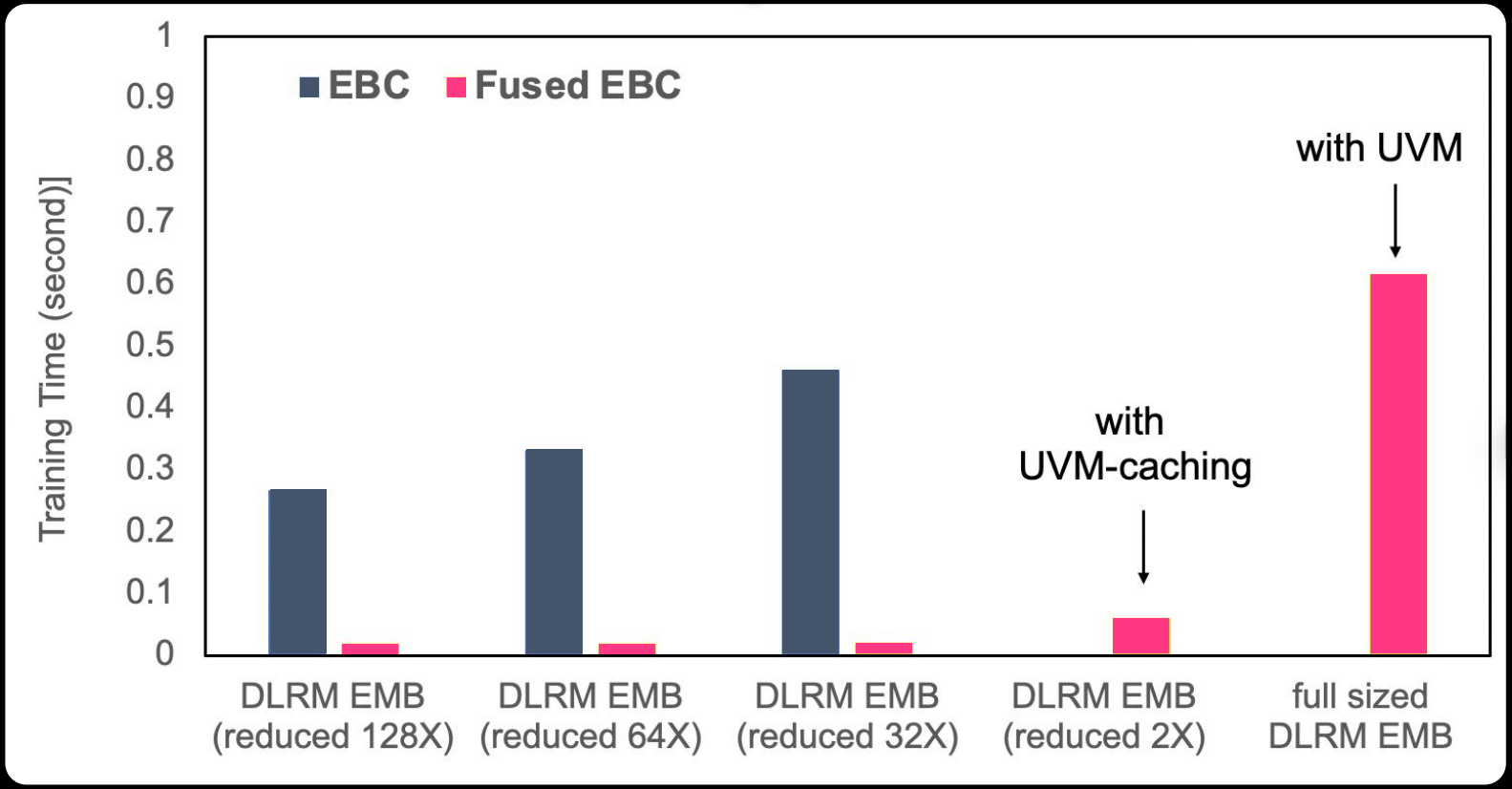
https://www.youtube.com/watch?v=QiGrTvsCZmA

Метрики по АБ

Действие (RU)	Метрика	Новизна	Сред. эффек	Нужно ещё дней
[DWH]Сессия главной ленты	time	1	1.96%	0
[DWH]Медианная продолжительность сессии главной ленты	time	1	2.02%	0
[DWH] Живые деньги с рекламы в ленте	counts	1	2.89%	0
[DWH]Лайк	counts	1	5.20%	0
[DWH] Живые деньги с рекламы в ленте	money	1	2.19%	0

Выводы

С помощью упомянутых оптимизаций становится возможным обучать огромные нейросетевые модели в целях рекомендательных систем



https://github.com/pytorch/torchrec/tree/main/benchmarks

Если захотите повторить

- Упомянутые оптимизации для обучения модели, в том числе с использованием библиотеки TorchRec
- Hashing Trick
- Feature Store
- Налаженный инференс на бекенде

https://notes.mail.ru/share/fc73f769-ca85-48e2-9f3d-63fc35f34243

t.me/mlvok

Спасибо!