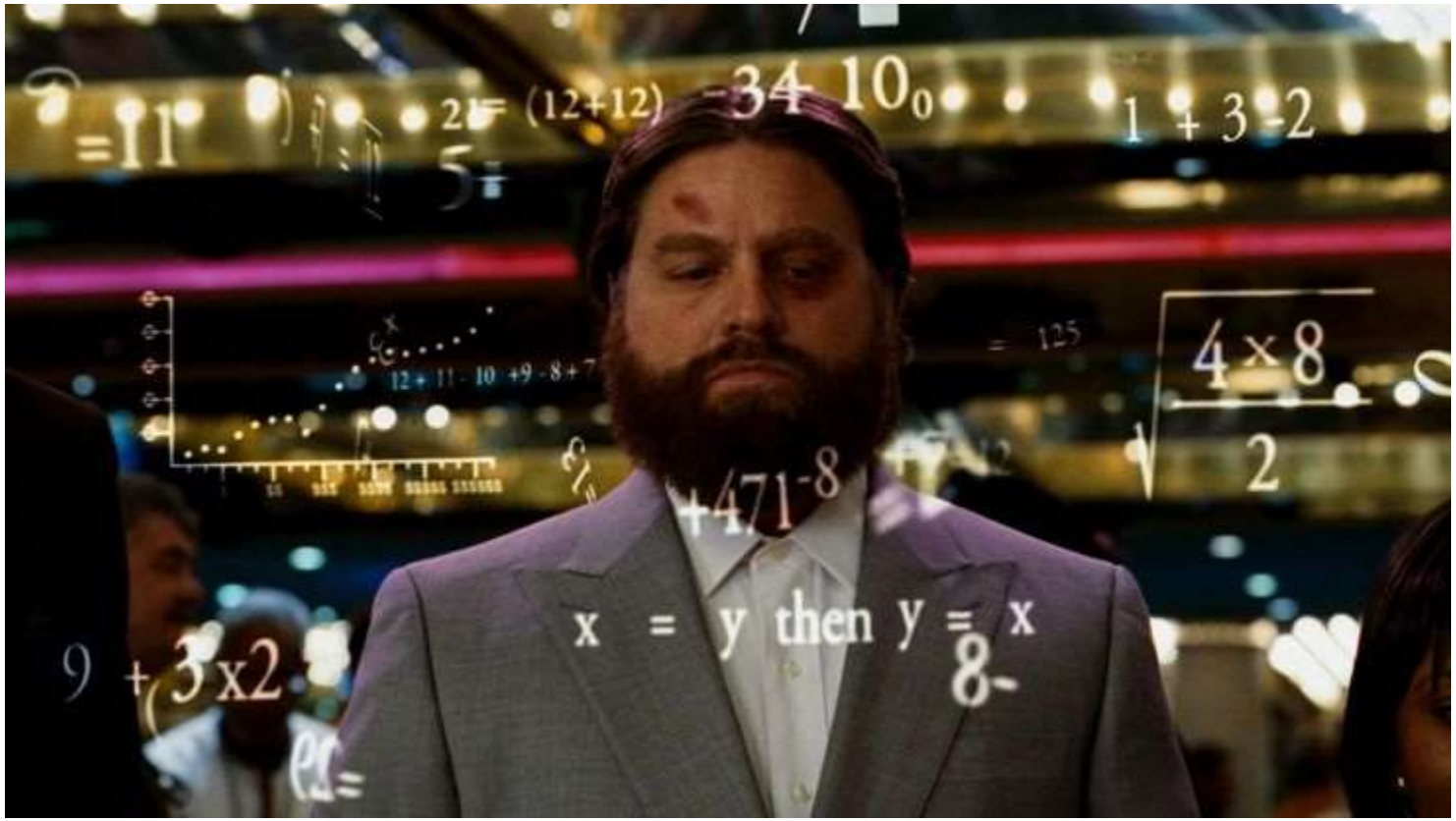
Эффективное обучение LLM

GPT-2 training memory footprint

1.5B параметров – на что уйдет память при обучении и сколько?



GPT-2 training memory footprint

На что уходит память?

- Параметры модели
- Градиенты
- Параметры оптимизатора
- Активации
- CUDA-контекст
- Данные в памяти

GPT-2 training memory footprint

Сколько?

- Параметры модели – **1.5B * 4 (fp32 = 4 байта) = 6GB**
- Градиенты - **1.5B * 4 (fp32 = 4 байта) = 6GB**
- Параметры оптимизатора (Adam) – **2 * 1.5B * 4 байта = 12GB**
- Активации:
  - **12** * hidden_dim * sequence_length * batch_size * num_layers * precision
  - **12** *1600 * 1000 * 16 * 48 * 4 = **59GB**
- CUDA-контекст **~1GB**
- Данные в памяти – совсем мало

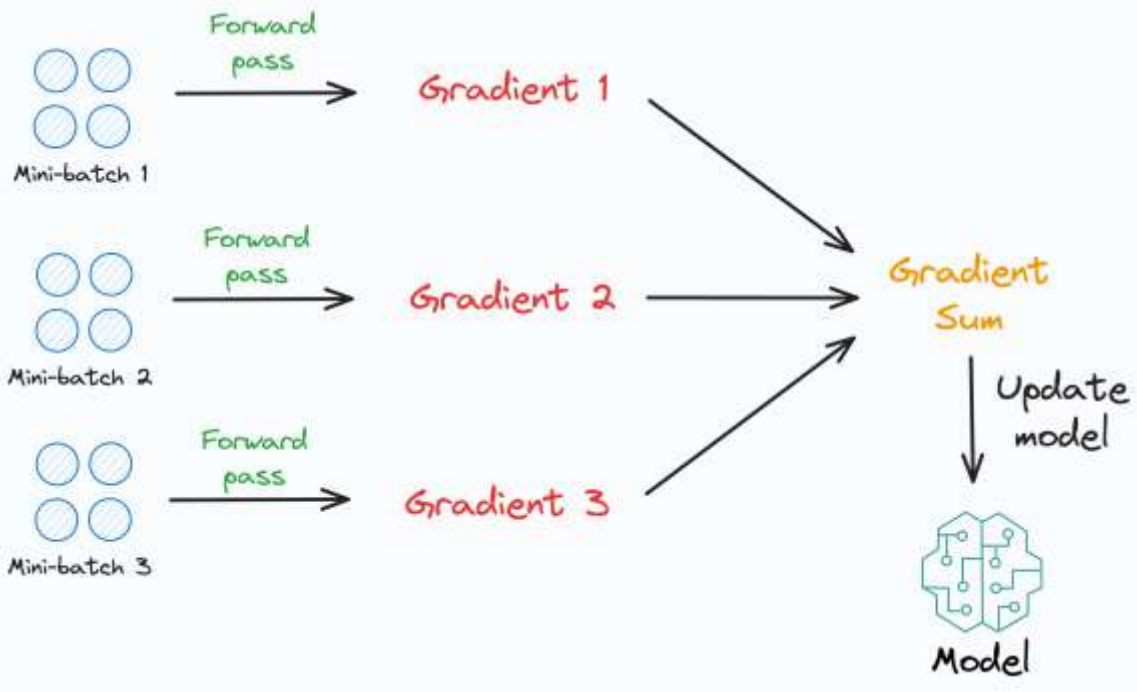
Итого: **~84GB**

GPT-2 training memory footprint

Как оптимизировать?

- **Уменьшить batch-size** (линейно влияет на активации)
- **Уменьшить seq len** (линейно влияет на активации)
- **AMP** – уменьшает память на активации вдвое ускоряет обучение
- **Gradient checkpointing** – снижает память на активации, замедляет обучение
- **Квантизация***
- **Offload на CPU**
- **PEFT**
- **Распределенное обучение**

Gradient accumulation




```python
for epoch in range(num_epochs):
    train_acc = torchmetrics.Accuracy(
        task="multiclass", num_classes=2).to(fabric.device)

    for batch_idx, batch in enumerate(train_loader):
        model.train()

        ### FORWARD AND BACK PROP
        outputs = model(
            batch["input_ids"],
            attention_mask=batch["attention_mask"],
            labels=batch["label"])
                                                    New
        outputs["loss"] = outputs["loss"] / accumulation_steps
        fabric.backward(outputs["loss"])

        ### UPDATE MODEL PARAMETERS          New
        if not batch_idx % accumulation_steps:
            optimizer.step()
            optimizer.zero_grad()

        ### LOGGING
        model.eval()
        with torch.no_grad():
            predicted_labels = torch.argmax(outputs["logits"], 1)
            train_acc.update(predicted_labels, batch["label"])
```

Automatic Mixed Precision

AMP автоматически определяет, какие операции следует выполнять в FP16, а какие — в FP32. Операции разделяются на три категории:

- Безопасные для FP16: линейные слои, свертки — выполняются в половинной точности для максимальной скорости
- Требующие FP32: операции редукции (сумма, среднее), softmax, loss функции — остаются в полной точности для стабильности
- Смешанные операции: автоматически приводятся к нужному типу данных

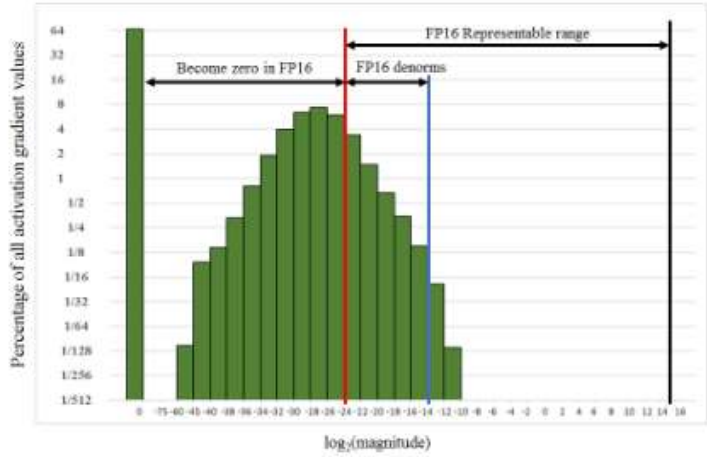
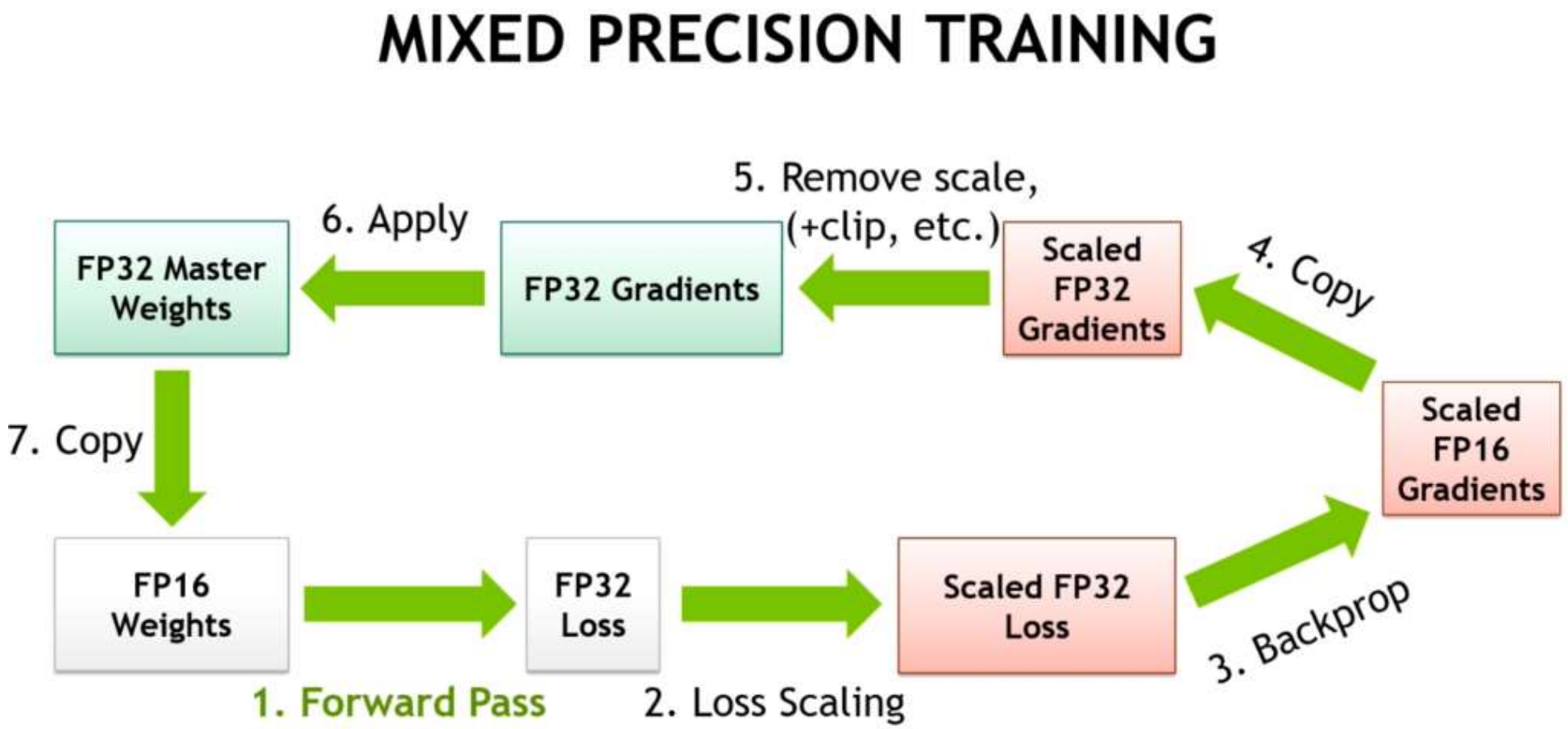


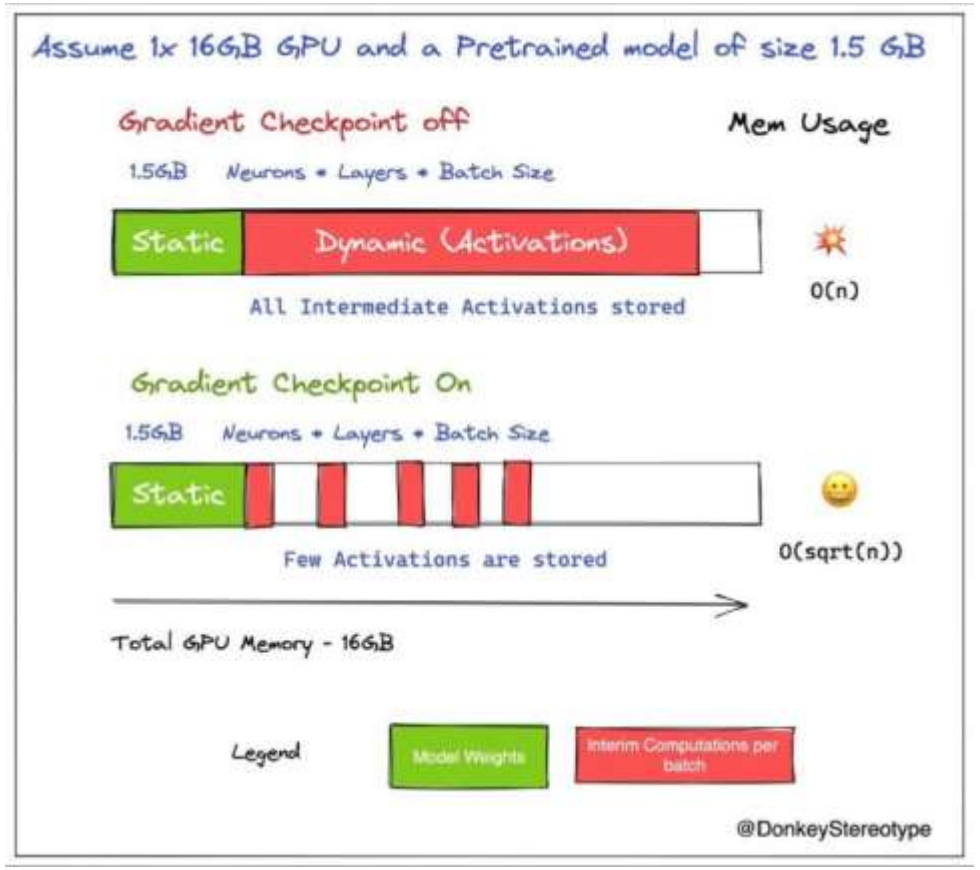
Figure 3: Histogram of activation gradient values during the training of Multibox SSD network. Note that the bins on the x-axis cover varying ranges and there's a separate bin for zeros. For example, 2% of the values are in the $[2^{-34}, 2^{-32})$ range, 2% of values are in the $[2^{-24}, 2^{-23})$ range, and 67% of values are zero.

Automatic Mixed Precision



Gradient checkpointing

Сеть разбивается на сегменты, и вместо сохранения всех активаций, сохраняются только активации определенных слоев (чекпойнтов)



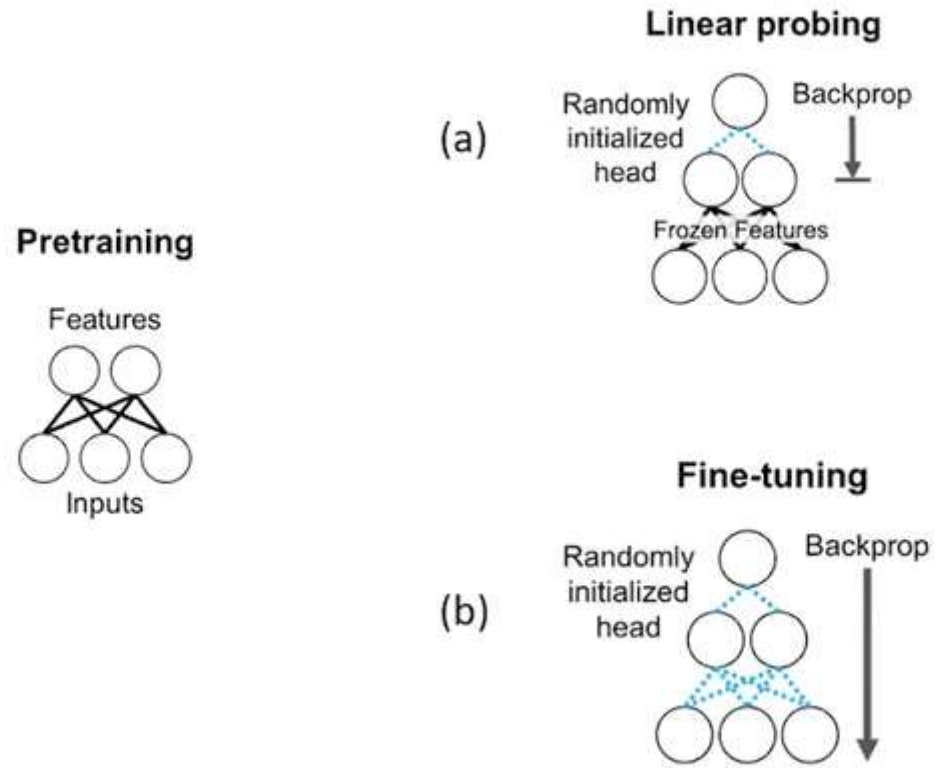
GPT-2 training memory footprint – **v2**

Сколько после оптимизаций?

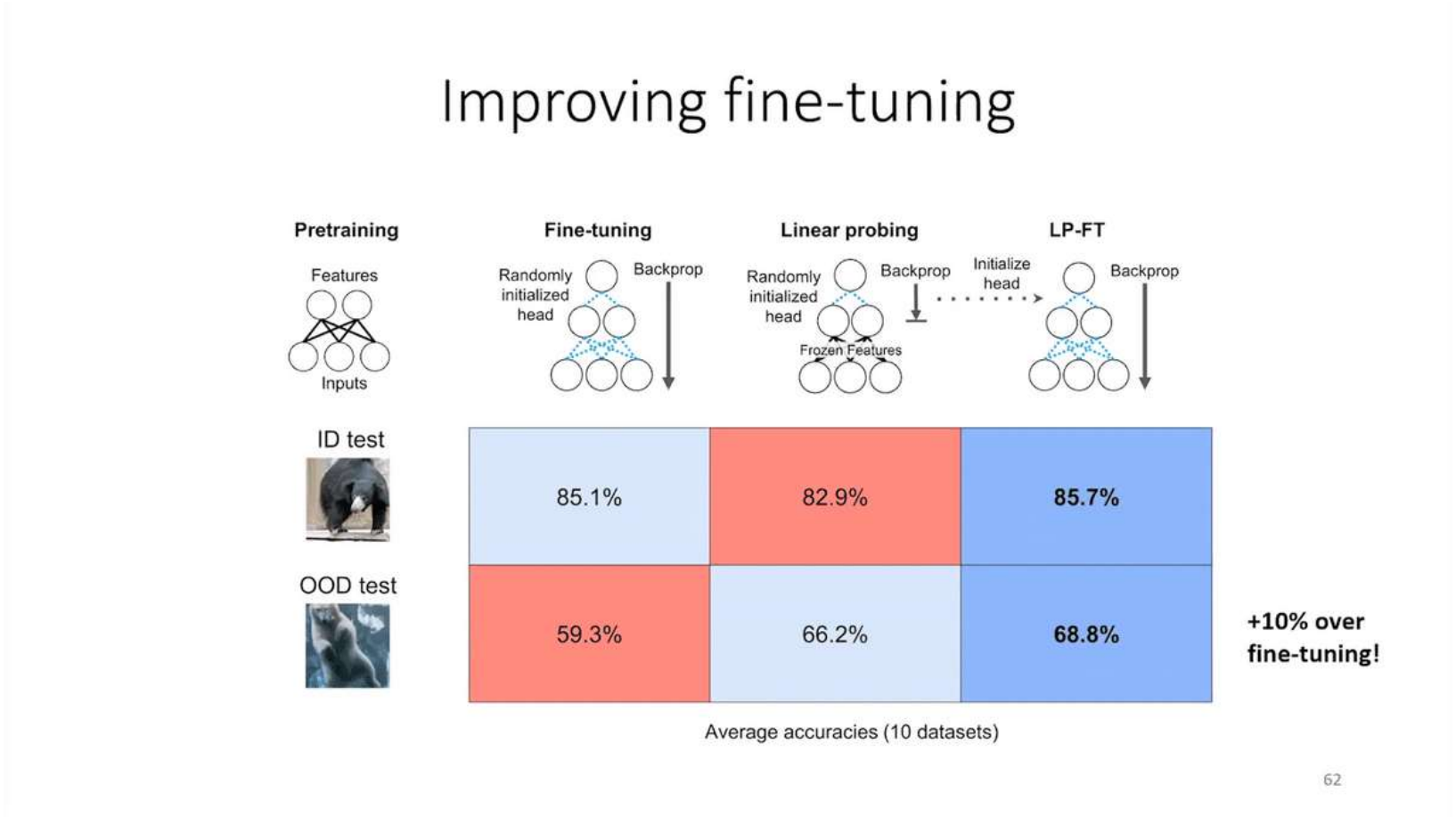
- Параметры модели – **1.5B * 2 (fp16 = 2 байта) = 3GB**
  - **+ мастер-копия весов в fp32 = 6GB**
- Градиенты - **1.5B * 2 (fp16 = 2 байта) = 3GB**
- Параметры оптимизатора (Adam) – **2 * 1.5B * 4 байта = 12GB**
- Активации:
  - **6** * hidden_dim * sequence_length * batch_size * num_layers * precision
  - **6** * 1600 * 1000 * 2 * 48 * 2 = **1.8GB**
- CUDA-контекст **~1GB**
- Данные в памяти – совсем мало

Итого: **26.8GB**

Linear probing



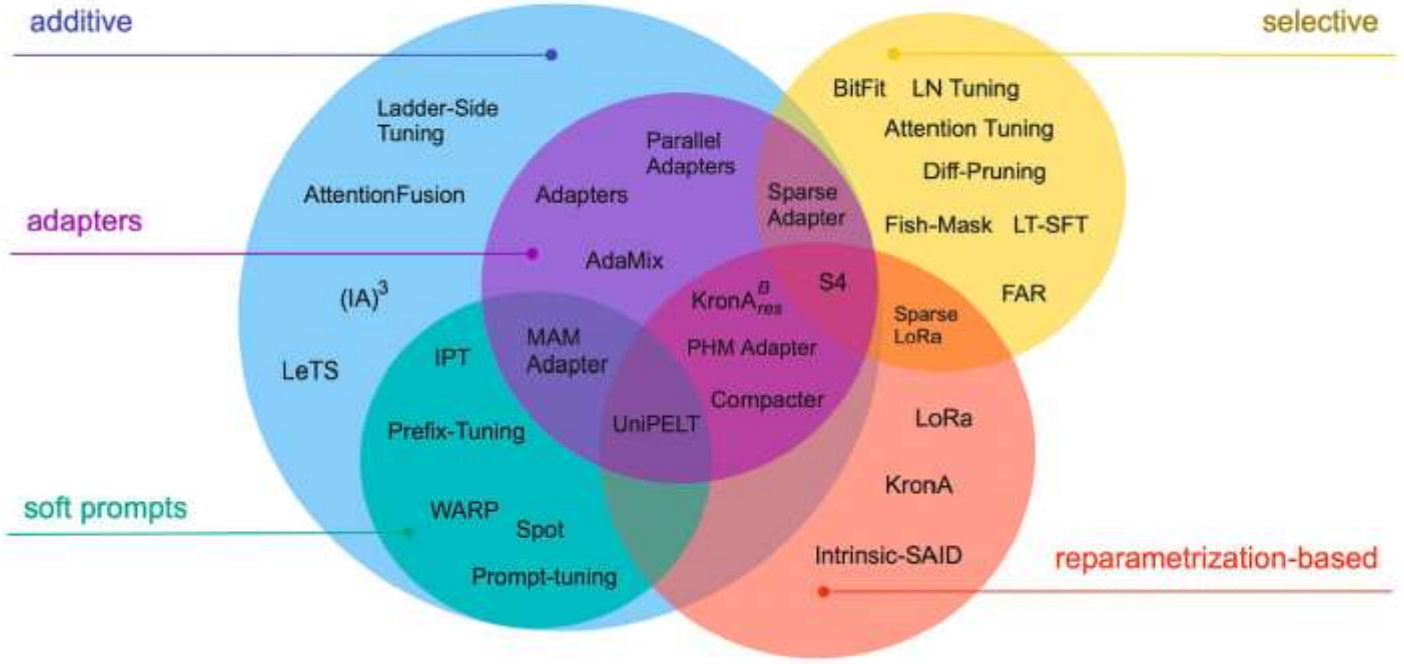
LP-FT



[Fine-Tuning can Distort Pretrained Features and Underperform Out-of-Distribution]

Разнообразие PEFT методов

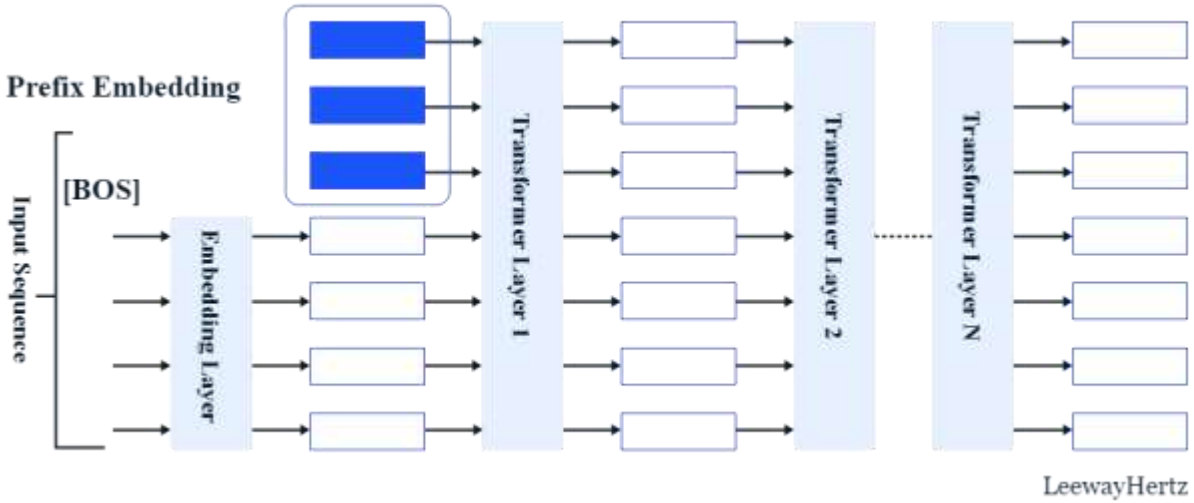
- **Адаптерные методы** - добавляют слои между слоями исходной модели
- **Методы низкоранговой адаптации** – добавляют обучаемые матрицы низкоранговой декомпозиции в слои исходной модели
- **Методы на основе промптов** – добавляют обучаемые эмбеддинги в разные места*
- **Селективные методы** – обучаются только выбранные параметры



PEFT – Prompt Tuning

```
1) "Translate the English sentence '{english_sentence}' into German: {german_translation}"

2) "English: '{english_sentence}' | German: {german_translation}"

3) "From English to German: '{english_sentence}' -> {german_translation}"
```

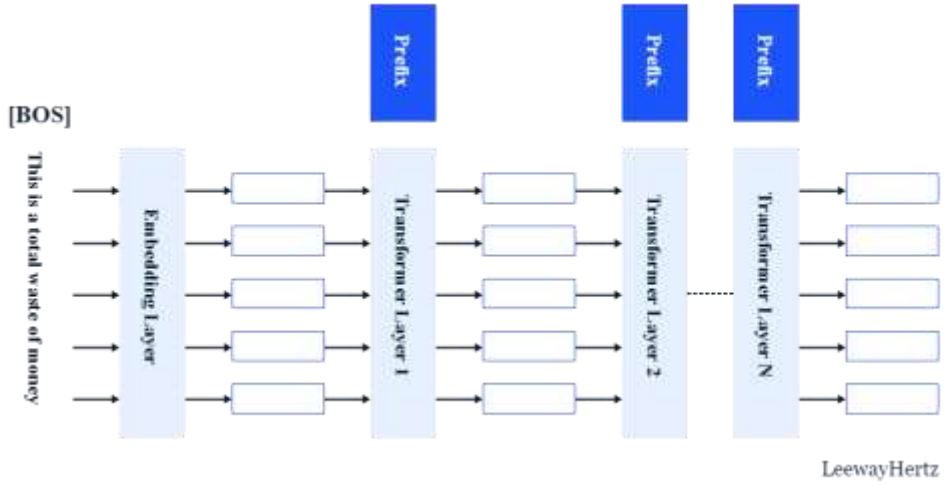
PEFT – Prompt Tuning



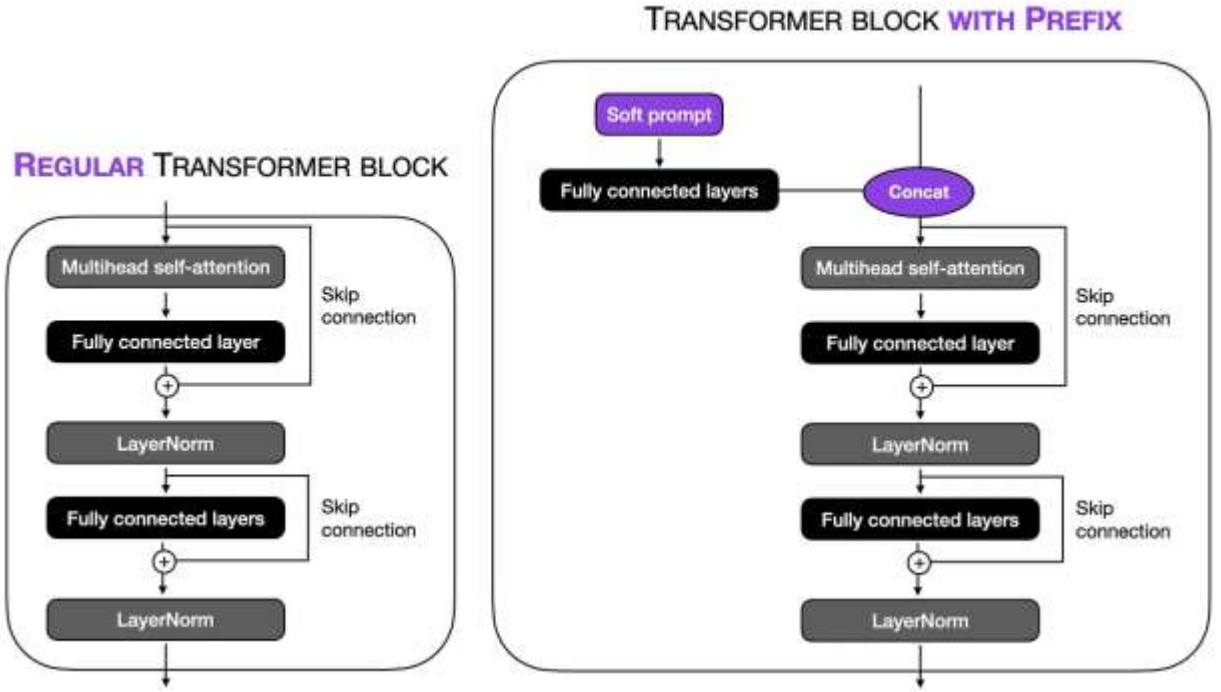

```python
soft_prompt = torch.nn.Parameter( # Make tensor trainable
    torch.rand(num_tokens, embed_dim)) # Initialize soft prompt tensor

def input_with_soft_prompt(x, soft_prompt) :
    x = concatenate([soft_prompt, x], # Prepend soft prompt to input
                    dim=seq_len)
    return x

# train soft prompt tensor via gradient descent
train(model(input_with_soft_prompt(x)))

# use model with soft prompts
model(input_with_soft_prompt(x))
```

[The Power of Scale for Parameter-Efficient Prompt Tuning]

PEFT – Prefix Tuning

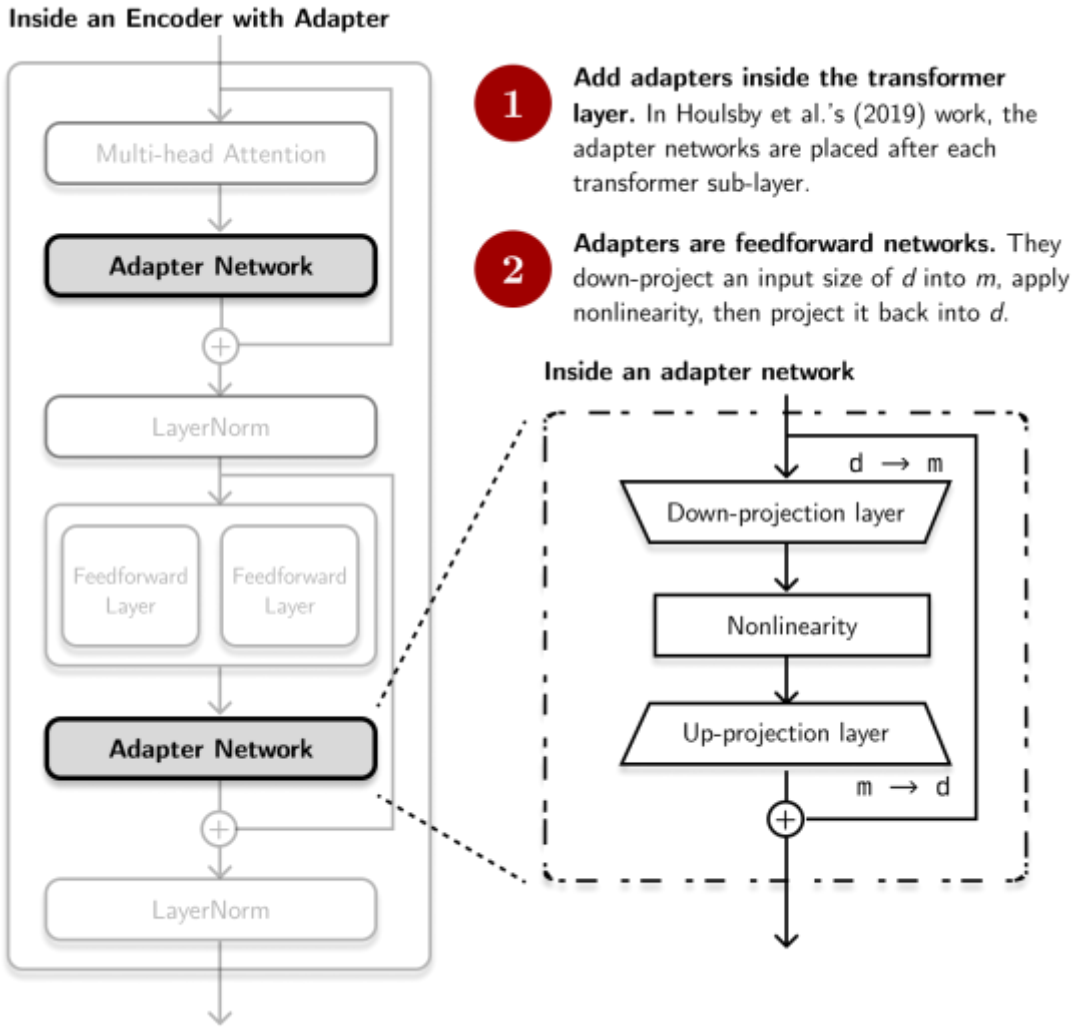



```
def transformer_block_with_prefix(x, soft_prompt):
    soft_prompt = FullyConnectedLayers(soft_prompt)  # Prefix
    x = concatenate([soft_prompt, x],                # Prefix
                    dim=seq_len)                     # Prefix
    residual = x
    x = self_attention(x)
    x = LayerNorm(x + residual)
    residual = x
    x = FullyConnectedLayers(x)
    x = LayerNorm(x + residual)
    return x
```

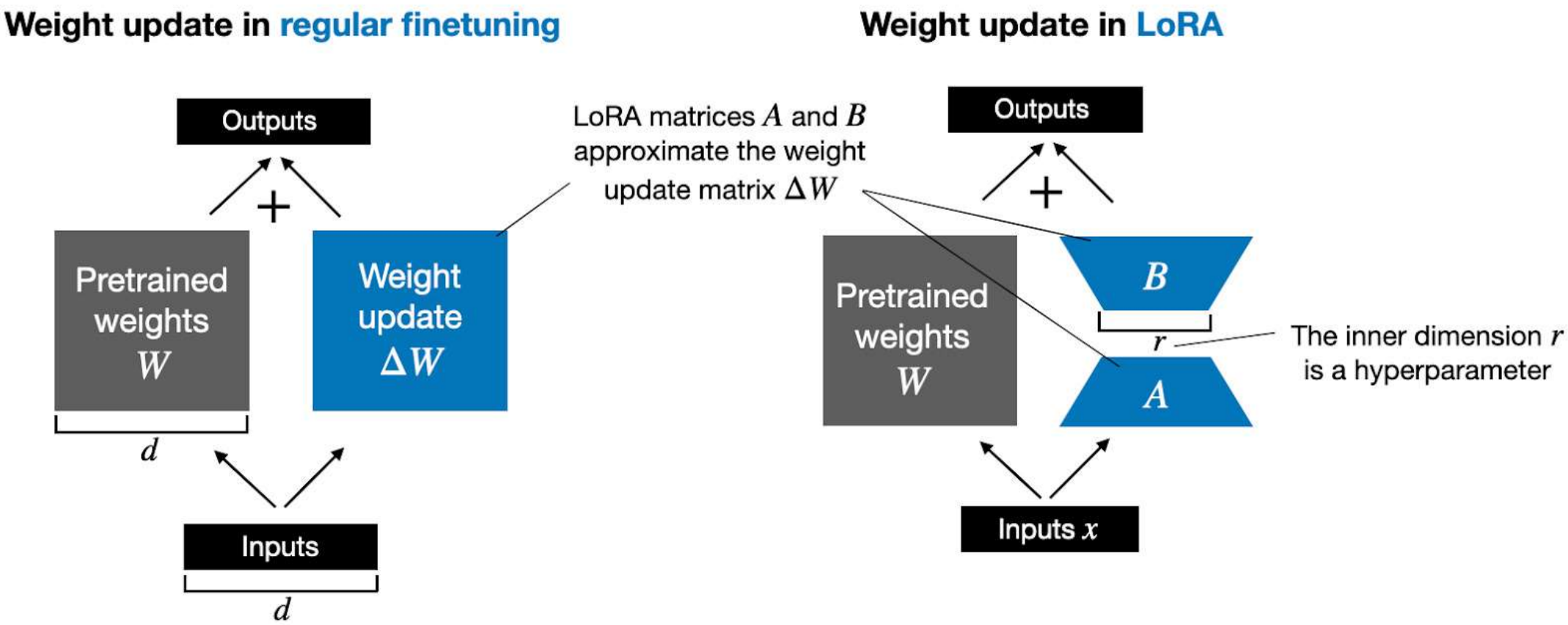


[Prefix-Tuning: Optimizing Continuous Prompts for Generation]

PEFT - Адаптеры

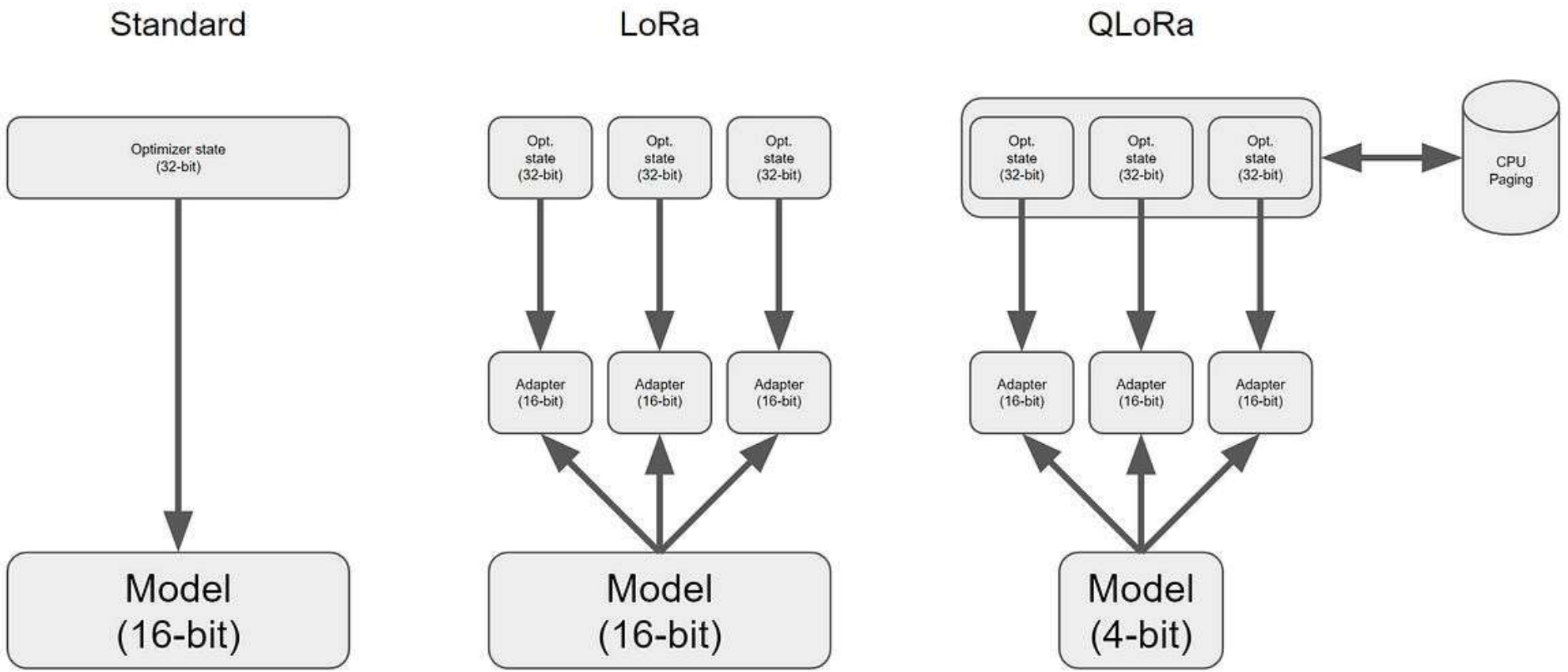


PEFT - LoRA



[LoRA: Low-Rank Adaptation of Large Language Models]

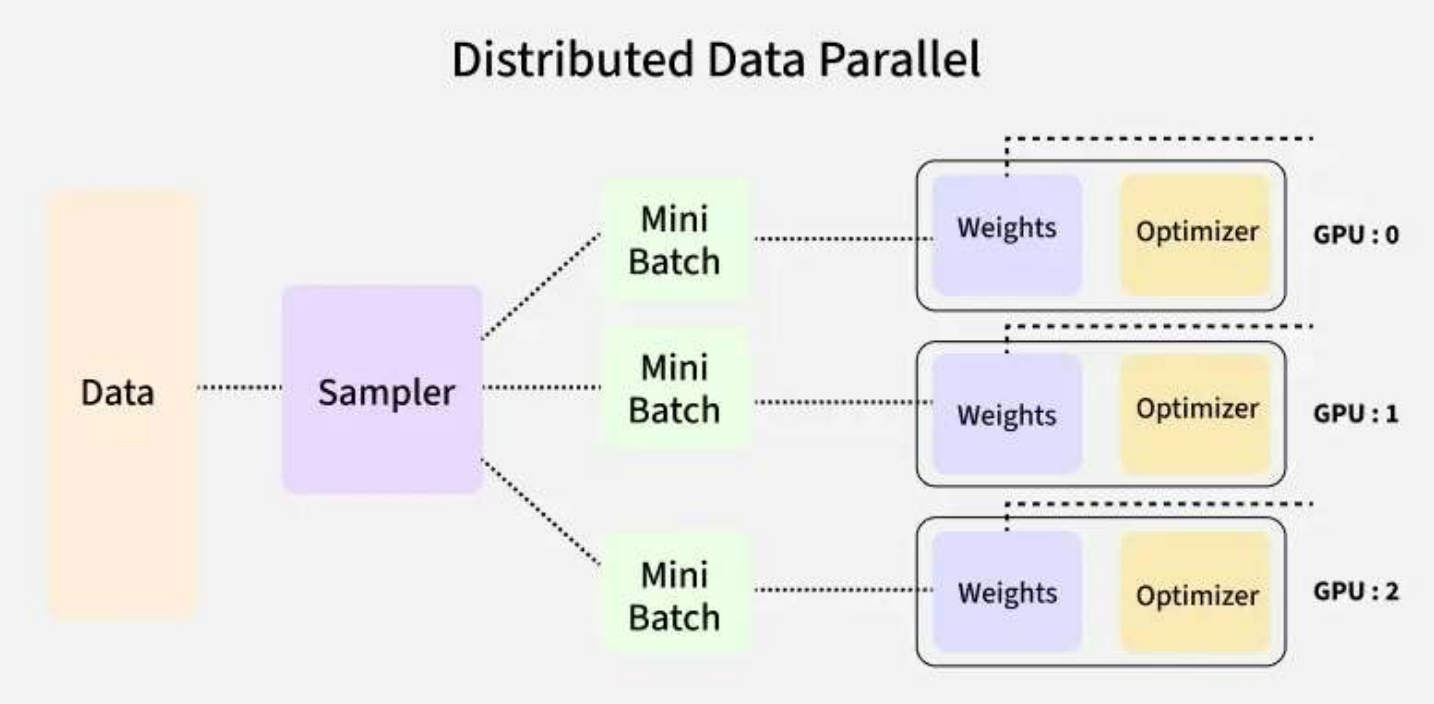
PEFT - QLoRA



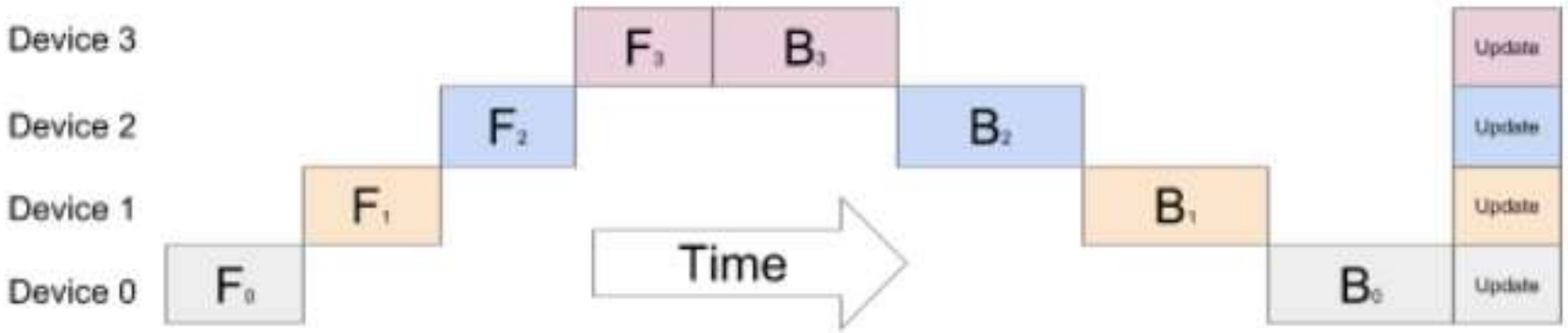
[QLoRA: Efficient Finetuning of Quantized LLMs]

Data Parallel

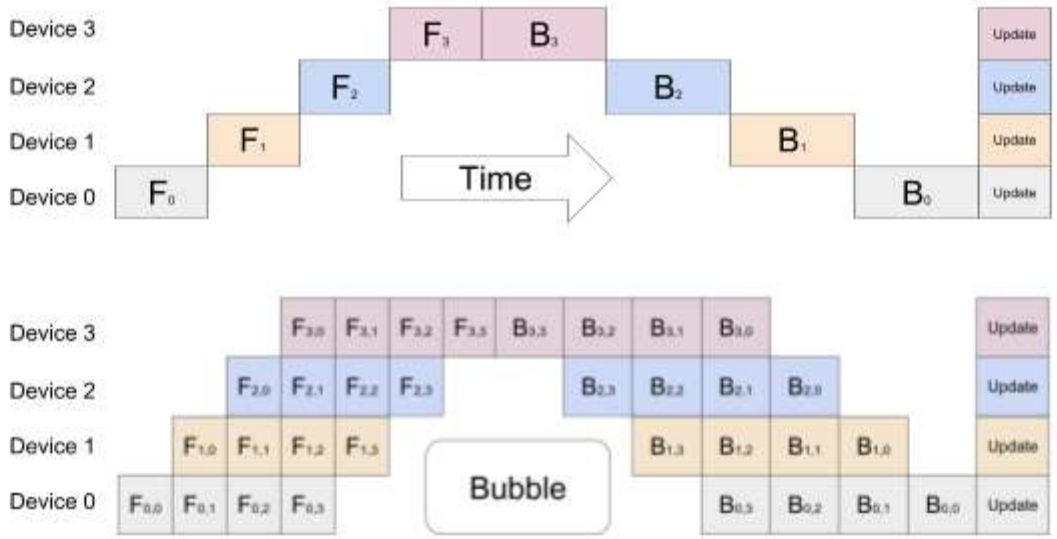
- Input разбивается и рассылается по GPU
- Параллельный FP и BP
- Градиенты собираются и усредняются на GPU0
- Параметры обновляются на GPU0 и копируются всем остальным
- Повторять до готовности



Naive model parallelism



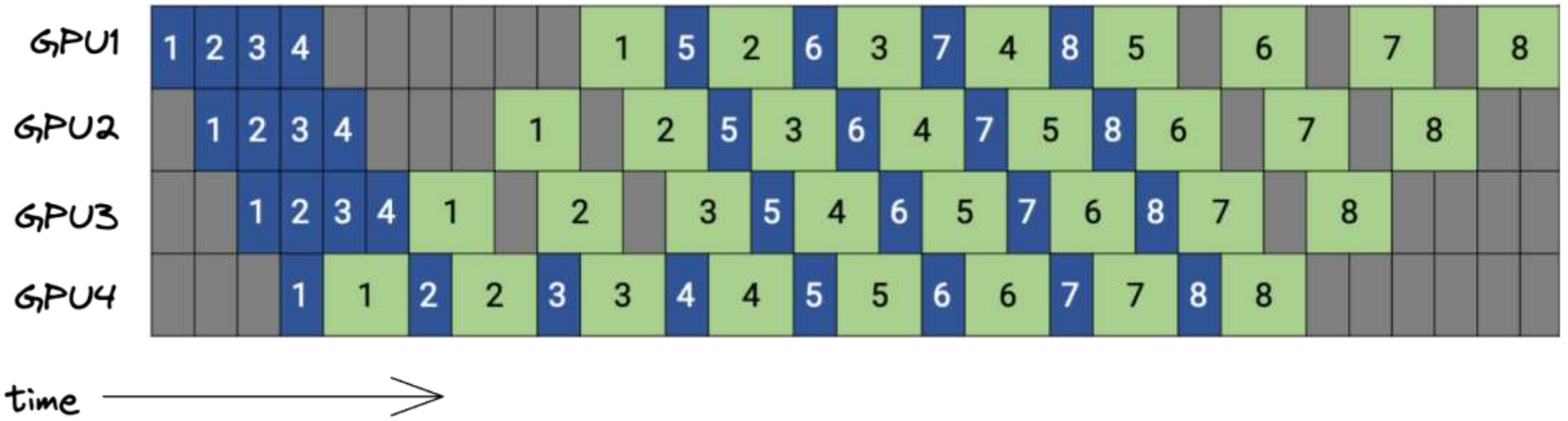
Pipeline model parallelism - GPipe



Top: The naive model parallelism strategy leads to severe underutilization due to the sequential nature of the network. Only one accelerator is active at a time. Bottom: GPipe divides the input mini-batch into smaller micro-batches, enabling different accelerators to work on separate micro-batches at the same time.

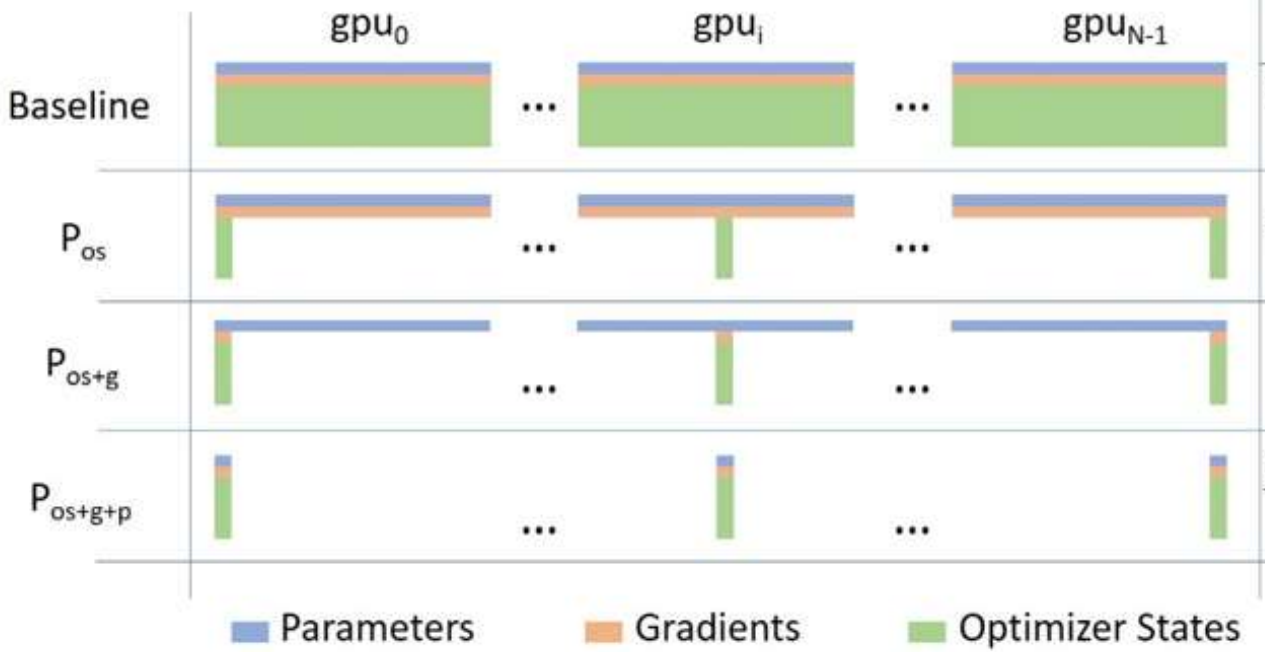
[Introducing GPipe, an Open Source Library for Efficiently Training Large-scale Neural Network Models]

Pipeline model parallelism - PipeDream



PipeDream: Fast and Efficient Pipeline Parallel DNN Training

Sharded training - ZeRO



	Memory Consumption		Comm Volume
	Formulation	Specific Example K=12 Ψ=7.5B N_d=64	
Baseline	$(2+2+K)*\Psi$	120GB	1x
P_{os}	$2\Psi+2\Psi+\frac{K*\Psi}{N_d}$	31.4GB	1x
P_{os+g}	$2\Psi+\frac{(2+K)*\Psi}{N_d}$	16.6GB	1x
P_{os+g+p}	$\frac{(2+2+K)*\Psi}{N_d}$	1.9GB	1.5x

[ZeRO: Memory Optimizations Toward Training Trillion Parameter Models]