Техники ускорения инференса и деплоя Large Language Models

30.08.2023

Саввов Сергей

За 7 лет в DS поработал в AdTech, FinTech, BioTech, trading и crypto-проектах.

Основал и руководил командой разработки торговых роботов с использованием ML.

Живу беспокойной жизнью номада и путешествую с 2-мя большими собаками.

sergey-savvov

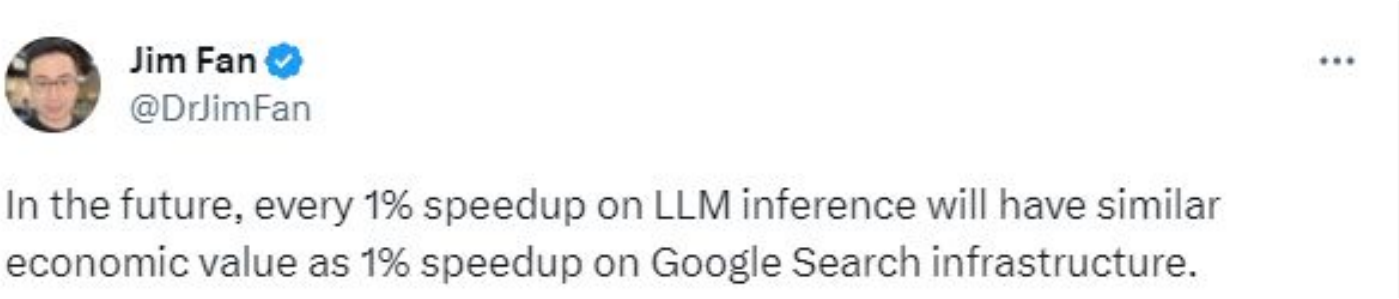
@slgero

Зачем нам это нужно?

Jim Fan
@DrJimFan

In the future, every 1% speedup on LLM inference will have similar economic value as 1% speedup on Google Search infrastructure.

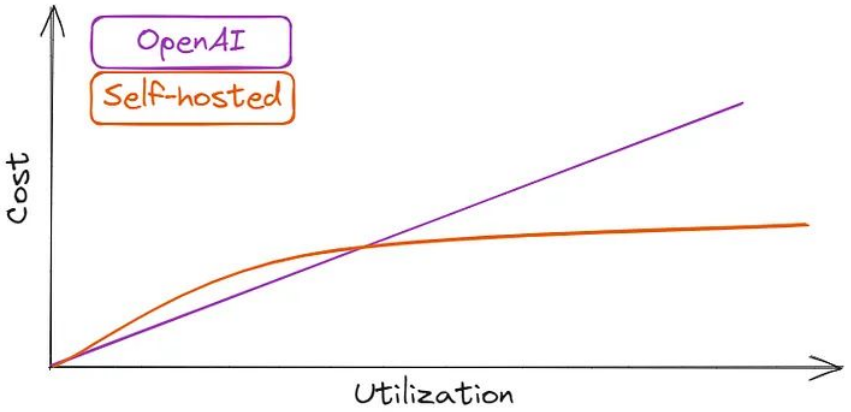
Зачем нам это нужно?



OpenAI тратят 700k$ ежедневно на хостинг ChatGPT:

Зачем нам ускорять?

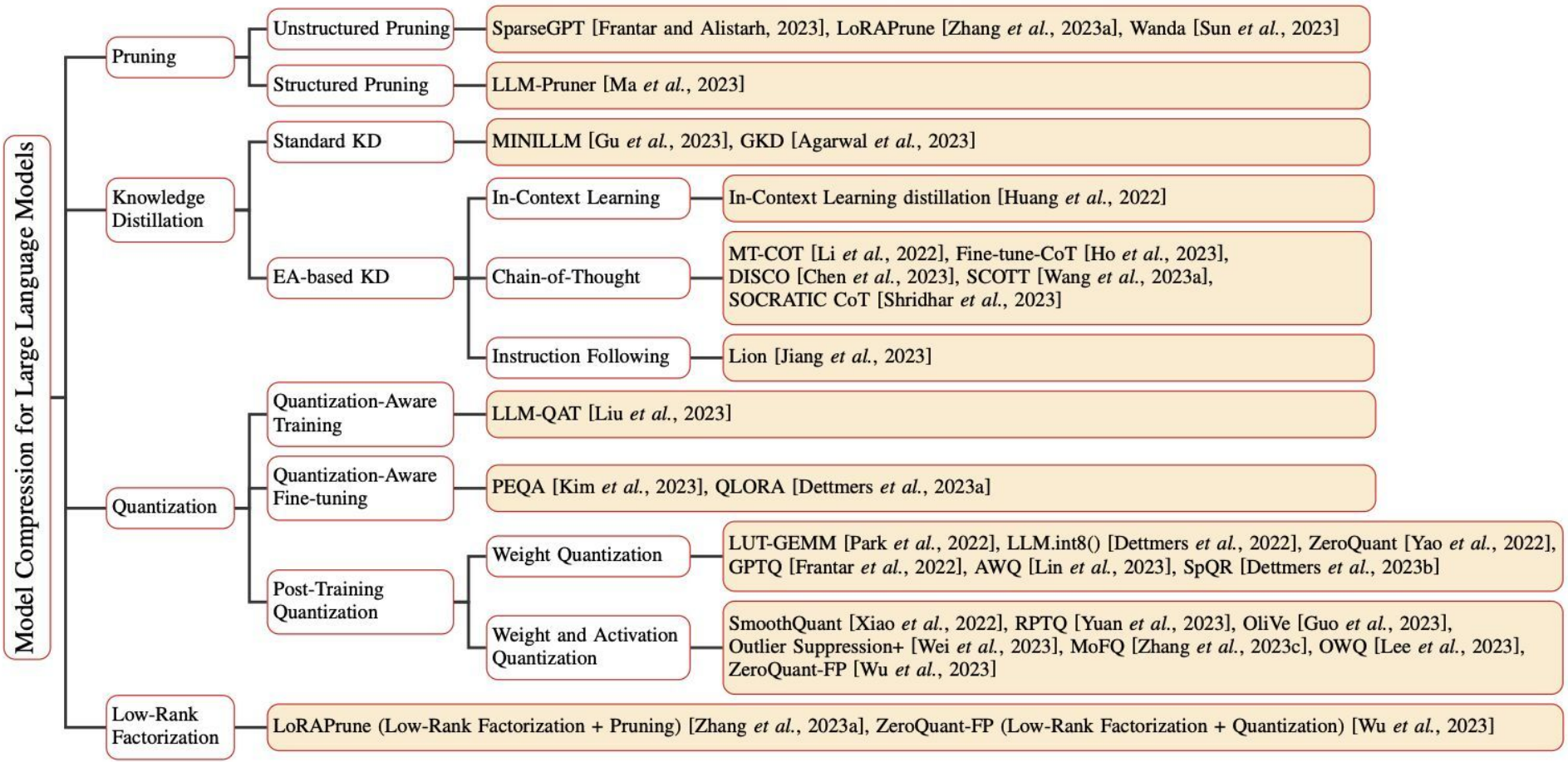
- Ускорение работы основного приложения
- Увеличение пропускной способности
- **Уменьшение трат на хостинг**
- **Запуск моделей на простых девайсах**
- Инженерный азарт



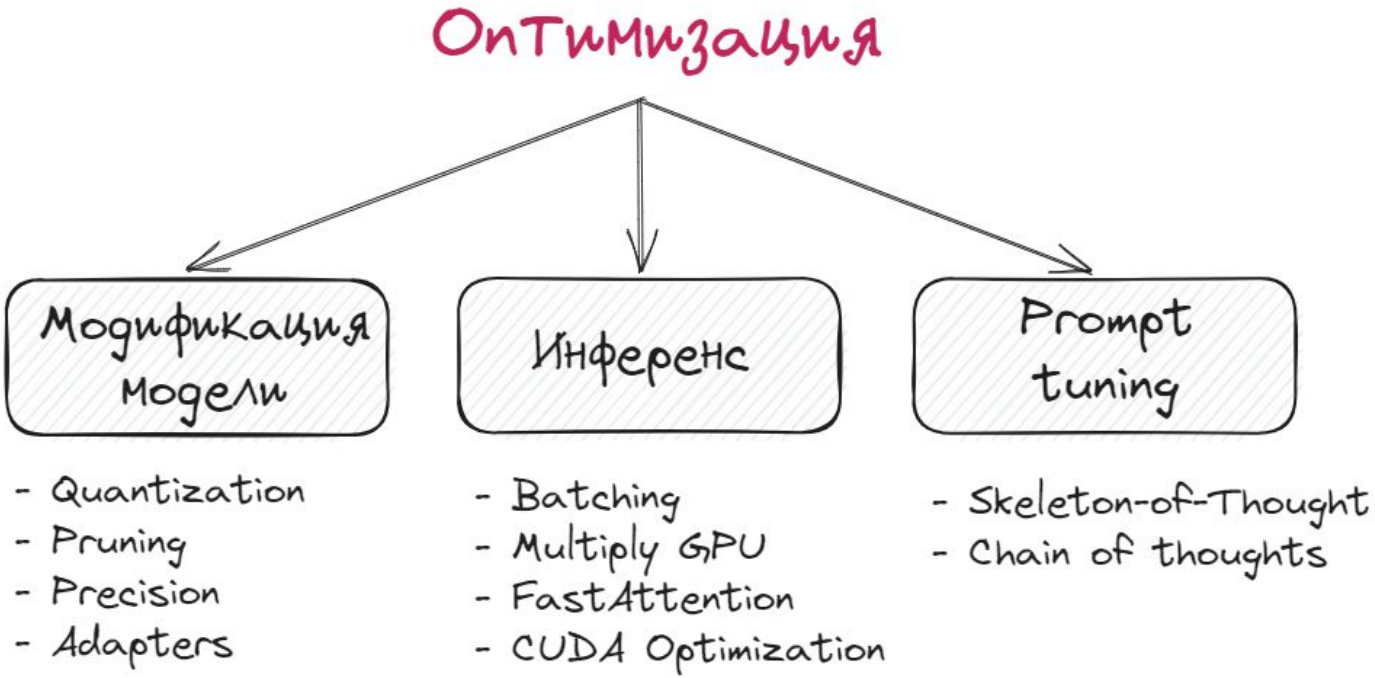
Схематичное сравнение стоимости OpenAI API и self-hosted LLM

Что ждать от доклада?

- Поговорим про разные техники ускорения и оптимизации: precision reduction, fine-tuning with adapters, continuous batching, ...
- Посмотрим на замеры скорости
- Узнаем, как имплементировать это в продакшене
- Пройдёмся по фреймворкам для деплоя


Model Compression for Large Language Models
Pruning
Unstructured Pruning
SparseGPT [Frantar and Alistarh, 2023], LoRAPrune [Zhang et al., 2023a], Wanda [Sun et al., 2023]
Structured Pruning
LLM-Pruner [Ma et al., 2023]
Knowledge Distillation
Standard KD
MINILLM [Gu et al., 2023], GKD [Agarwal et al., 2023]
EA-based KD
In-Context Learning
In-Context Learning distillation [Huang et al., 2022]
Chain-of-Thought
MT-COT [Li et al., 2022], Fine-tune-CoT [Ho et al., 2023], DISCO [Chen et al., 2023], SCOTT [Wang et al., 2023a], SOCRATIC CoT [Shridhar et al., 2023]
Instruction Following
Lion [Jiang et al., 2023]
Quantization
Quantization-Aware Training
LLM-QAT [Liu et al., 2023]
Quantization-Aware Fine-tuning
PEQA [Kim et al., 2023], QLORA [Dettmers et al., 2023a]
Post-Training Quantization
Weight Quantization
LUT-GEMM [Park et al., 2022], LLM.int8() [Dettmers et al., 2022], ZeroQuant [Yao et al., 2022], GPTQ [Frantar et al., 2022], AWQ [Lin et al., 2023], SpQR [Dettmers et al., 2023b]
Weight and Activation Quantization
SmoothQuant [Xiao et al., 2022], RPTQ [Yuan et al., 2023], OliVe [Guo et al., 2023], Outlier Suppression+ [Wei et al., 2023], MoFQ [Zhang et al., 2023c], OWQ [Lee et al., 2023], ZeroQuant-FP [Wu et al., 2023]
Low-Rank Factorization
LoRAPrune (Low-Rank Factorization + Pruning) [Zhang et al., 2023a], ZeroQuant-FP (Low-Rank Factorization + Quantization) [Wu et al., 2023]

Типы оптимизаций



Подготовка к эксперименту

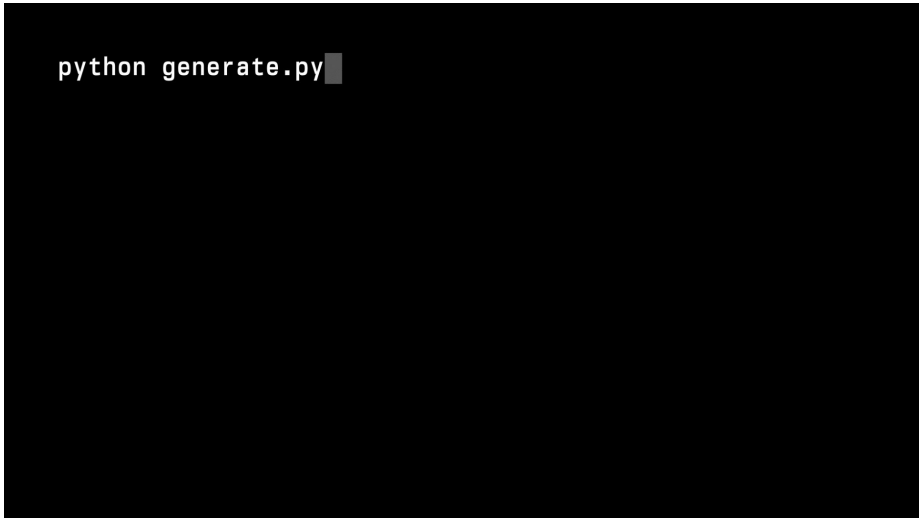
Модель: Falcon 7B

GPU: A100 40GB

Библиотека для экспериментов: Lit-GPT

- Можно проводить быстрые замеры
- Есть поддержка разных адаптеров
- Хорошо подходит для старта

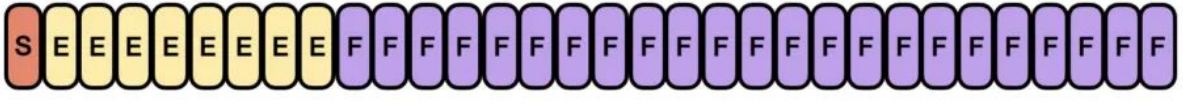
Github: https://github.com/Lightning-AI/lit-gpt



Модификация модели

Precision reduction

float 32



Sign (1 bit)

Exponent (8 bits)

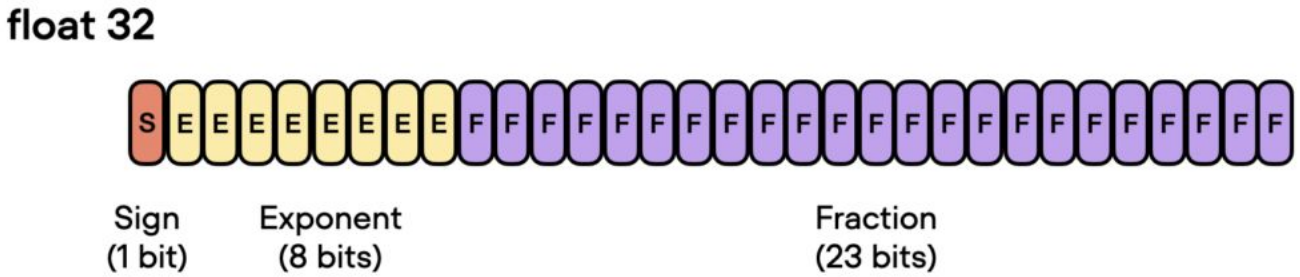
Fraction (23 bits)

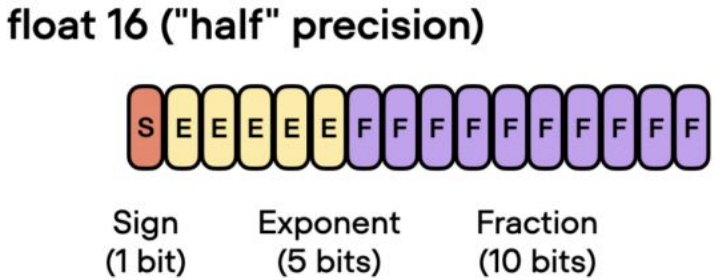
$$(-1)^{b_{31}} \times 2^{(b_{30}b_{29}\ldots b_{23})_2 - 127} \times (1.b_{22}b_{21}\ldots b_0)_2 \,,$$

which yields

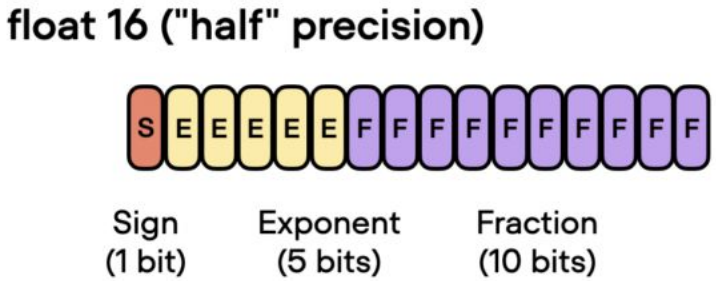
$$\text{value} = (-1)^{\text{sign}} \times 2^{(E-127)} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right).$$

Precision reduction



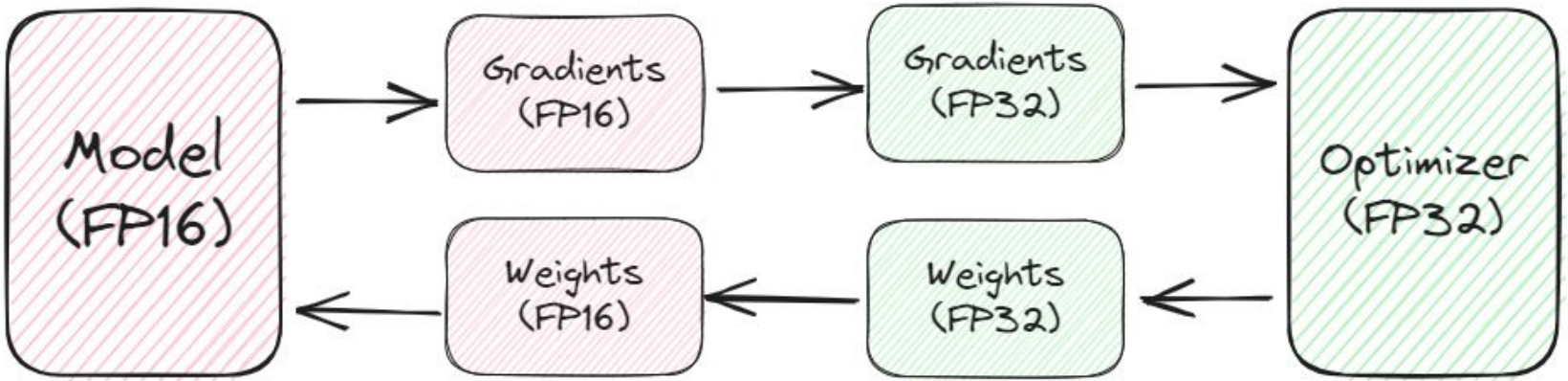


Precision reduction

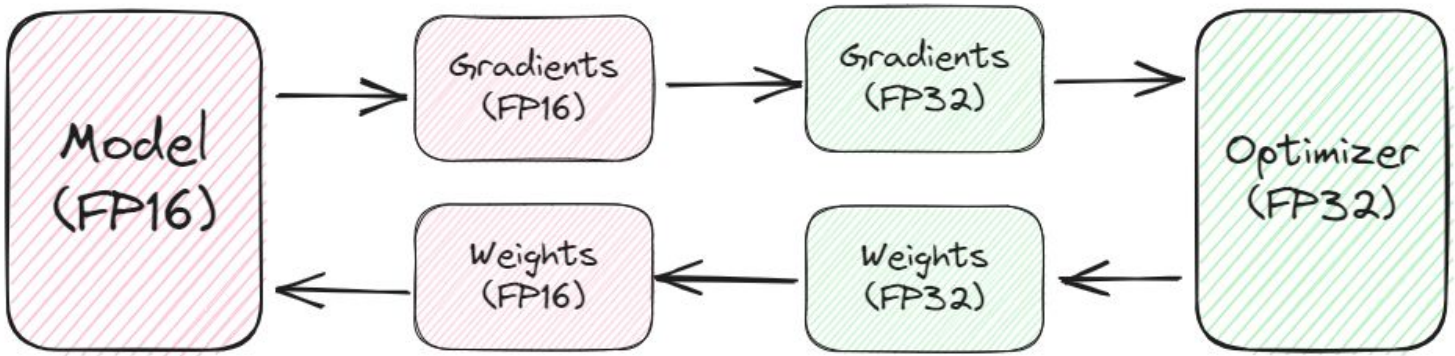



```
python generate/base.py \
    --prompt "I am so fast that I can" \
    --checkpoint_dir checkpoints/tiiuae/falcon-7b \
    --max_new_tokens 50 \
    --precision "16-true"
```

Mixed-precision



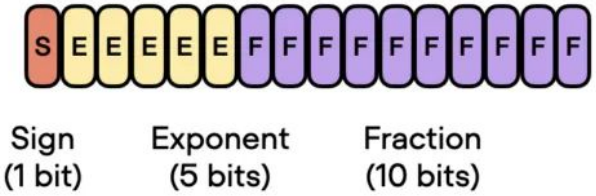
Mixed-precision



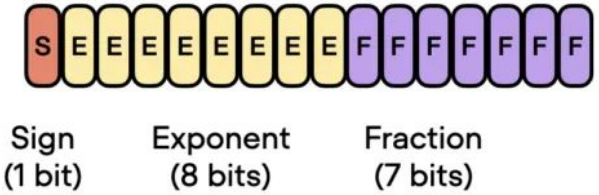

```
python generate/base.py \
    --prompt "I am so fast that I can" \
    --checkpoint_dir checkpoints/tiiuae/falcon-7b \
    --max_new_tokens 50 \
    --precision "16-mixed"
```

Brain floating point (Bfloat 16)

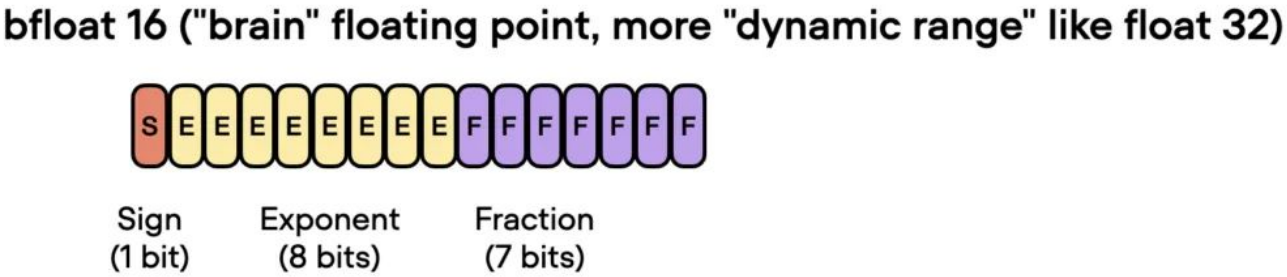
float 16 ("half" precision)



bfloat 16 ("brain" floating point, more "dynamic range" like float 32)



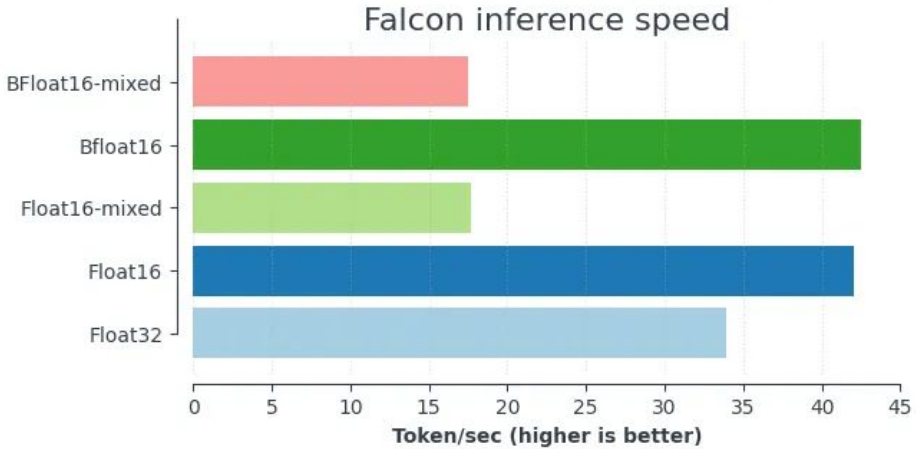
Brain floating point (Bfloat 16)

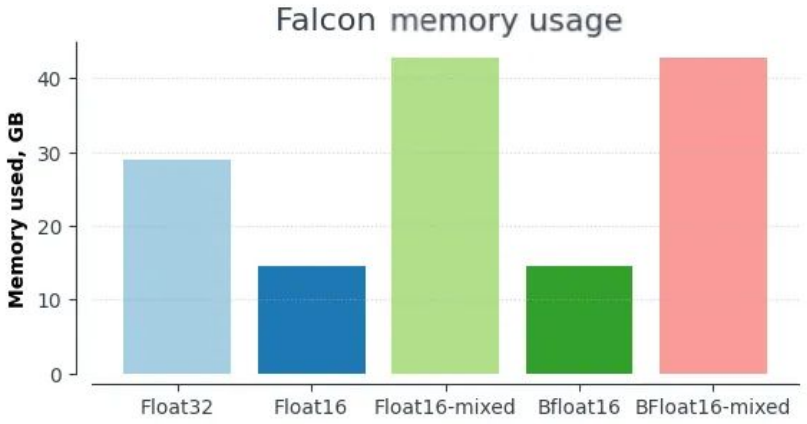



```
python generate/base.py \
    --prompt "I am so fast that I can" \
    --checkpoint_dir checkpoints/tiiuae/falcon-7b \
    --max_new_tokens 50 \
    --precision "bf16-true"
```

Хотя bfloat16 изначально был разработан для TPU, теперь этот формат поддерживается несколькими графическими процессорами NVIDIA.

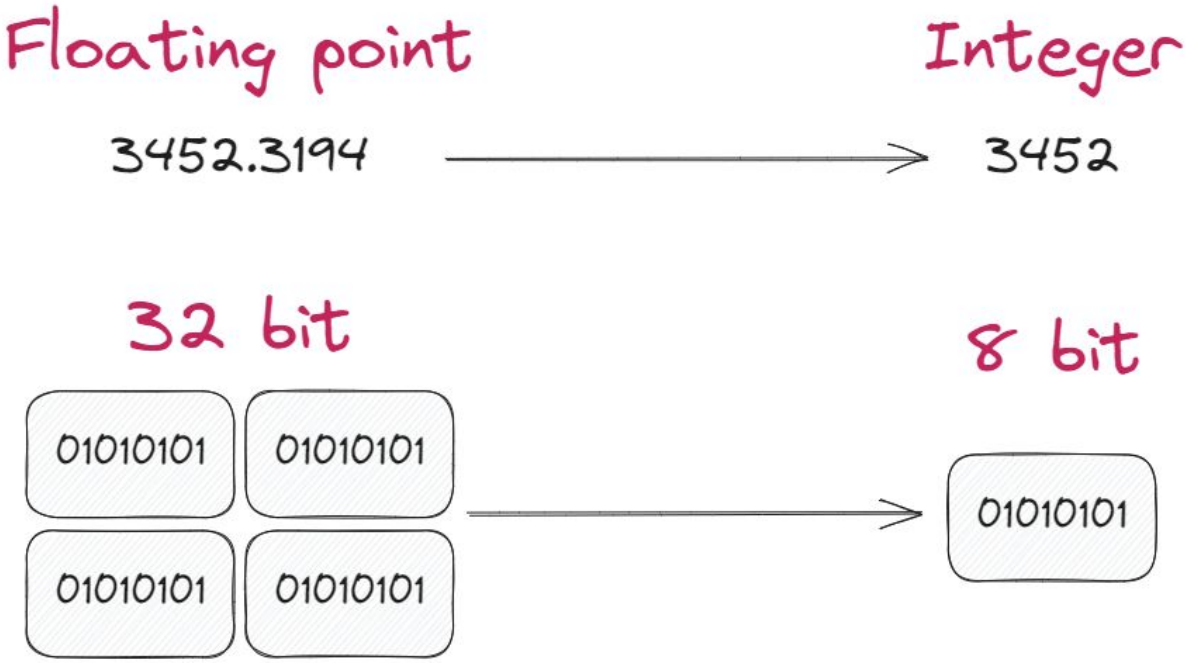
Precision reduction



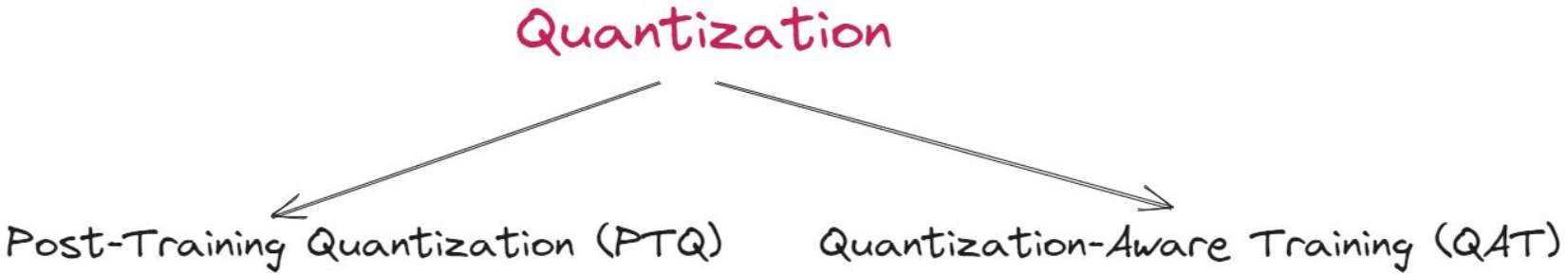


bfloat16 всех побеждает 🎉

Quantization



Quantization



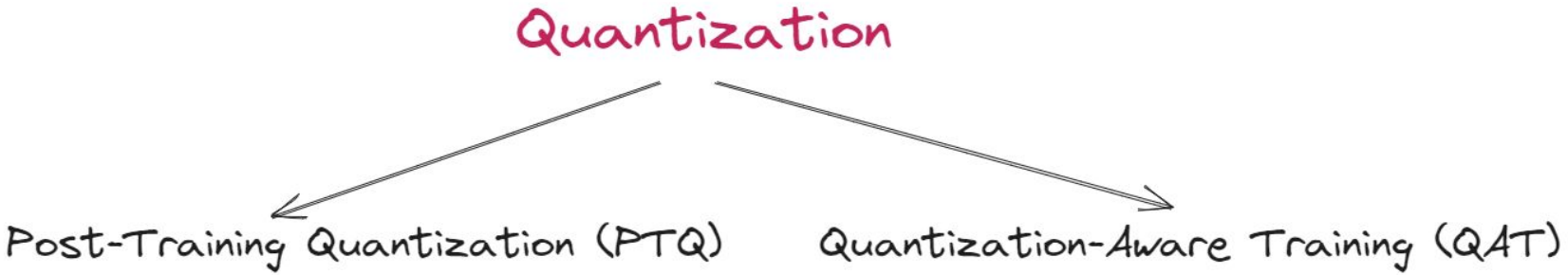
Самый “дешёвый” способ

Квантование применяется к УЖЕ обученной модели

Квантование применяется во время обучения, что заметно сложнее.

Качество модели обычно лучше

Quantization



Самый “дешёвый” способ

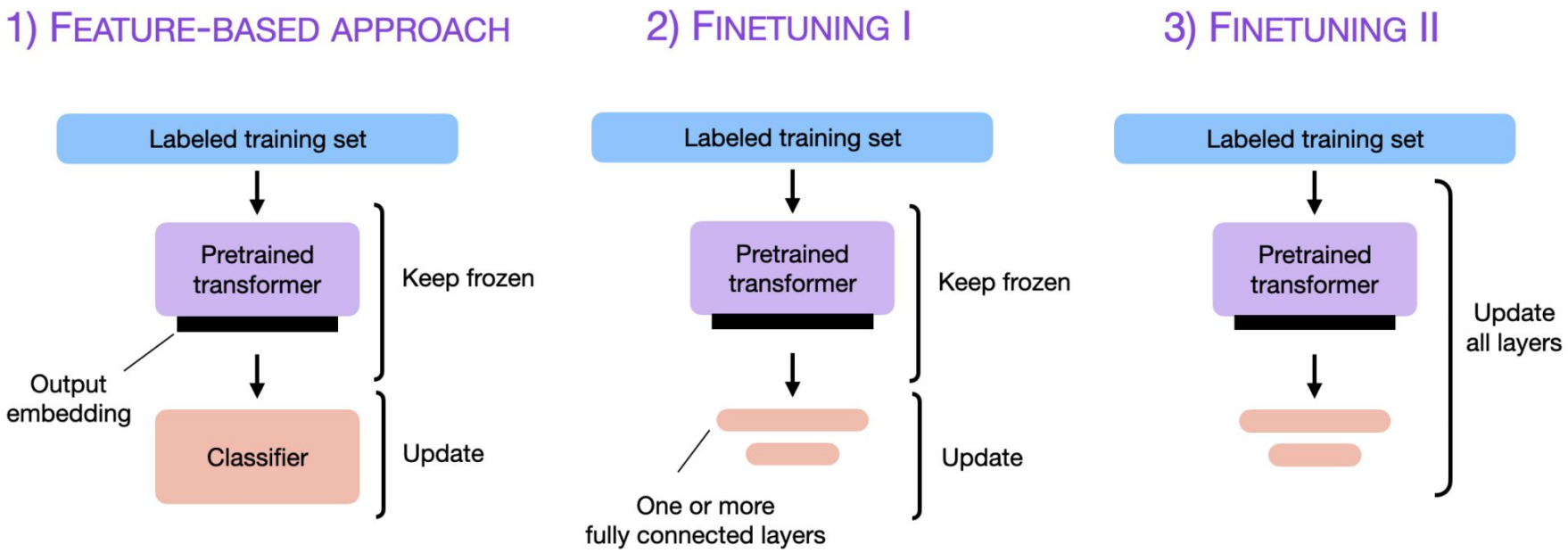
Квантование применяется к УЖЕ обученной модели

```
python generate.py \
    --prompt "I am so fast that I can" \
    --quantize llm.int8
```

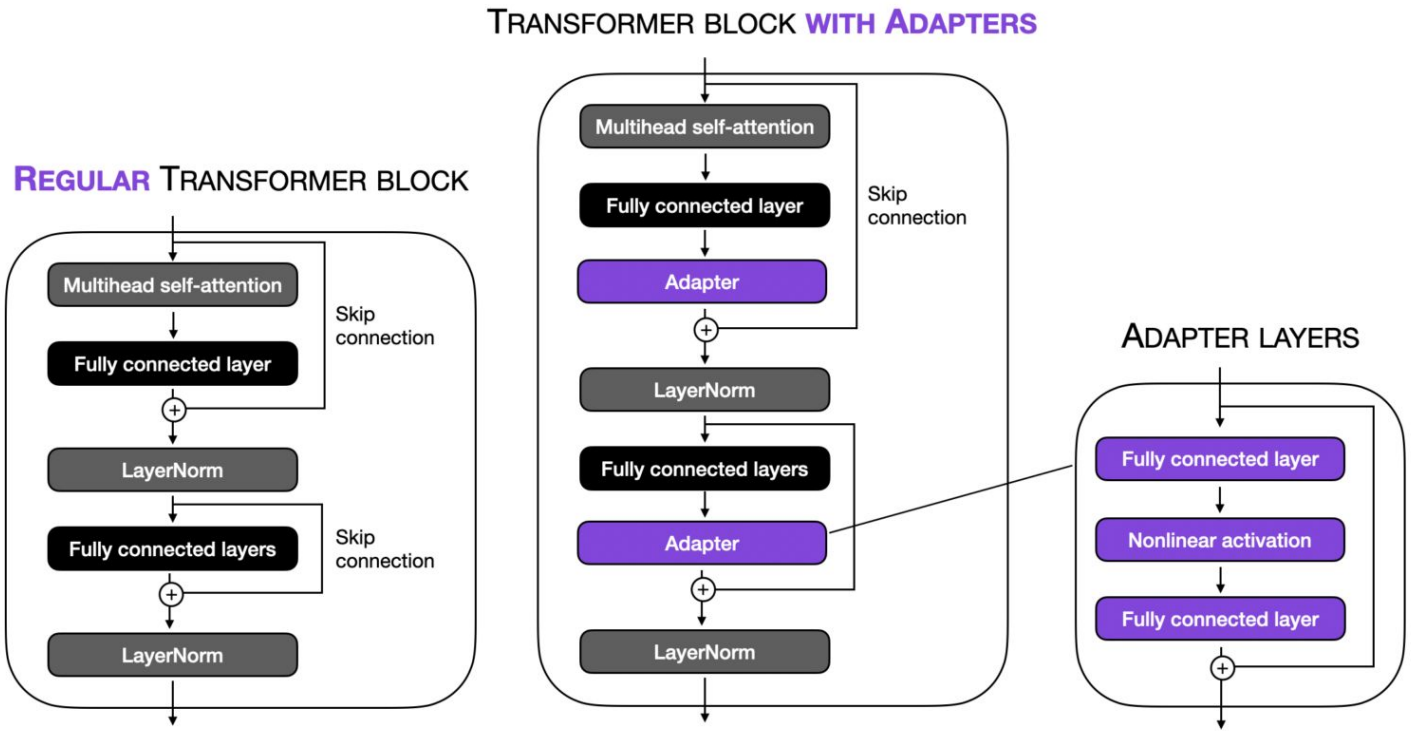
Квантование применяется во время обучения, что заметно сложнее.

Качество модели обычно лучше

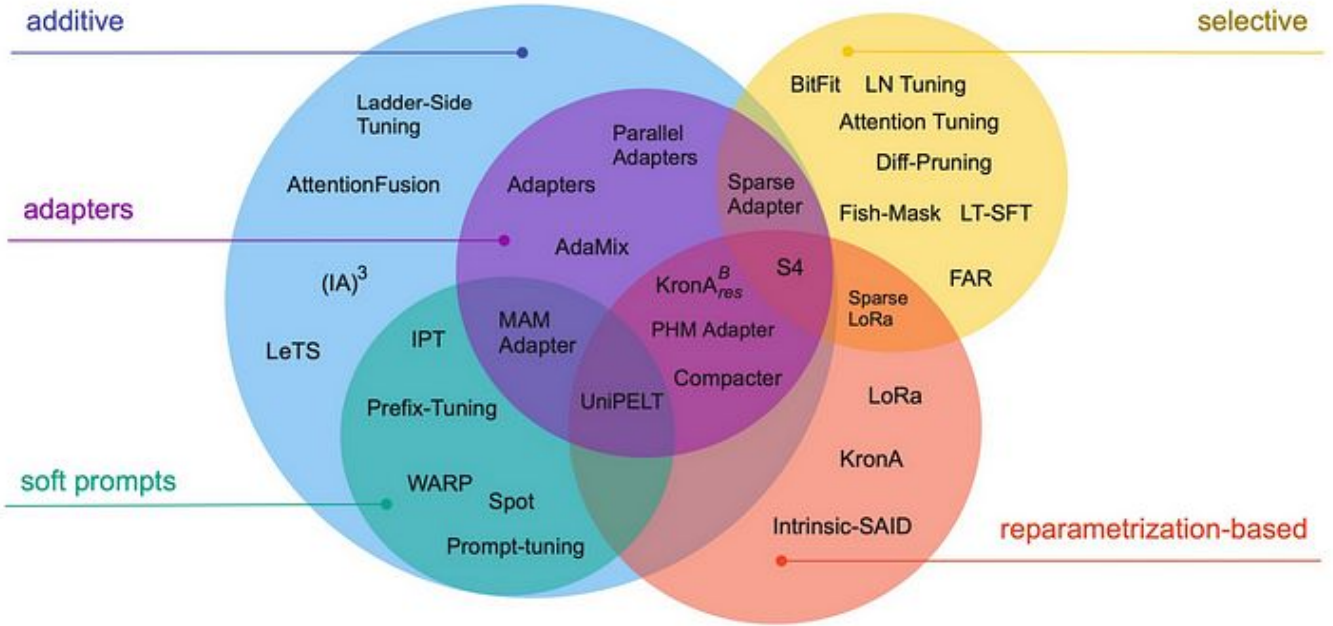
Fine-Tuning Before Adapters



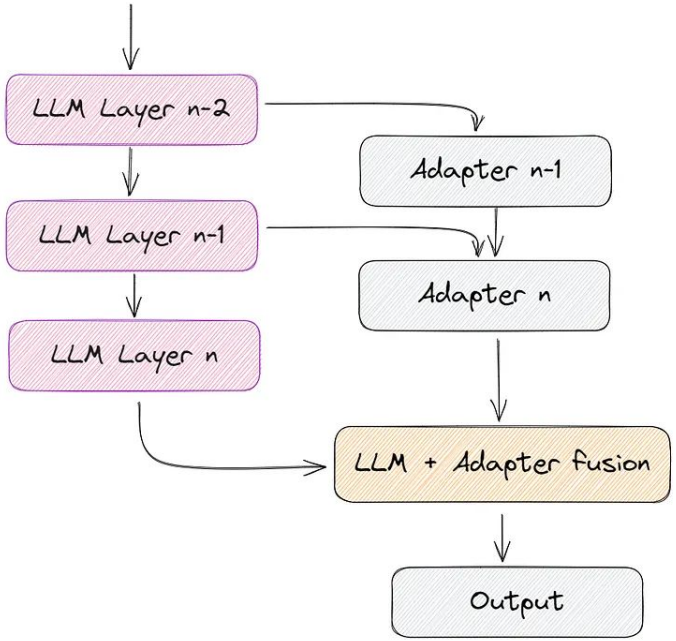
Fine-Tuning With Adapters



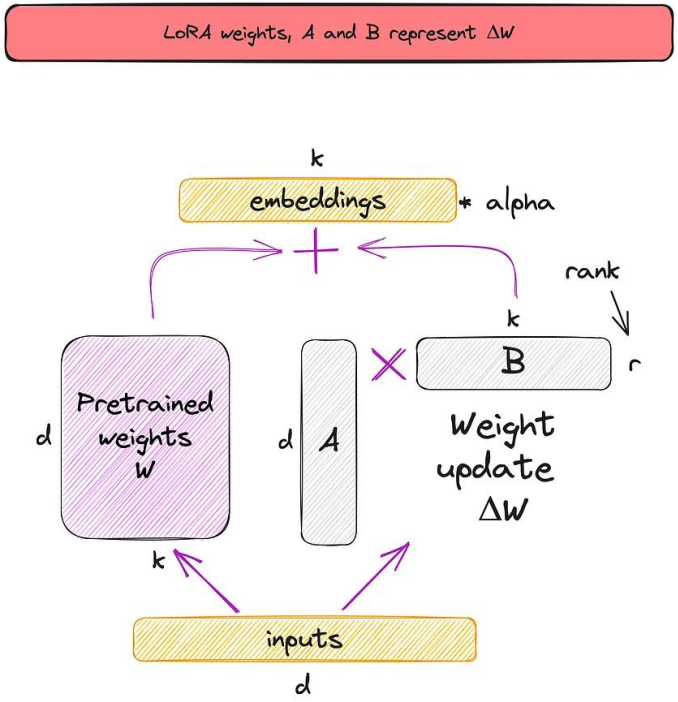
Adapters


additive
selective
adapters
soft prompts
reparametrization-based
Ladder-Side Tuning
AttentionFusion
Parallel Adapters
Adapters
AdaMix
Sparse Adapter
BitFit
LN Tuning
Attention Tuning
Diff-Pruning
Fish-Mask
LT-SFT
FAR
S4
Sparse LoRa
(IA)3
LeTS
IPT
MAM Adapter
PHM Adapter
Compacter
UniPELT
Prefix-Tuning
LoRa
KronA
WARP
Spot
Prompt-tuning
Intrinsic-SAID

LoRA

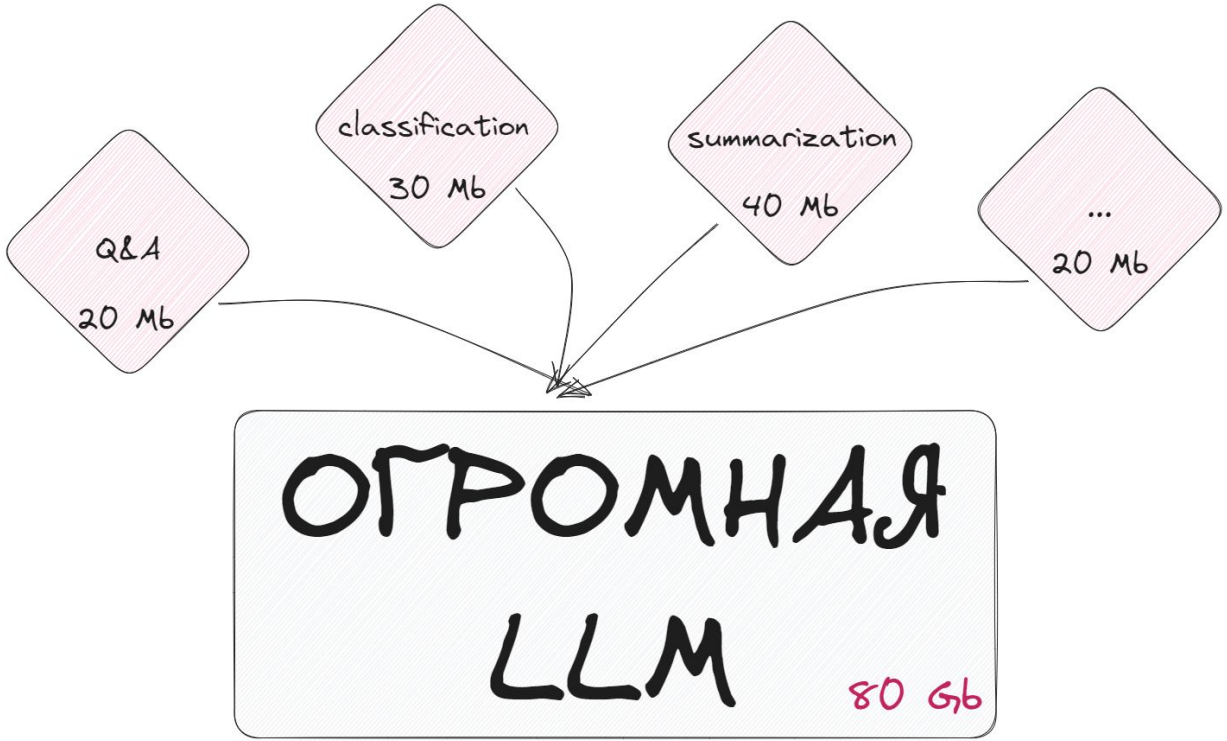


[Architecture of adapter-based knowledge injection into LLMs]



Forward pass with updated model.

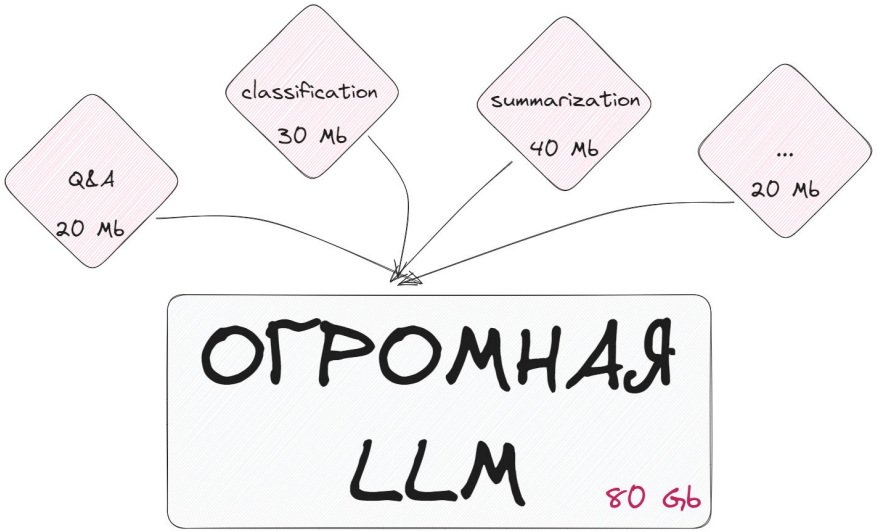
Fine-Tuning With Adapters



Fine-Tuning With Adapters

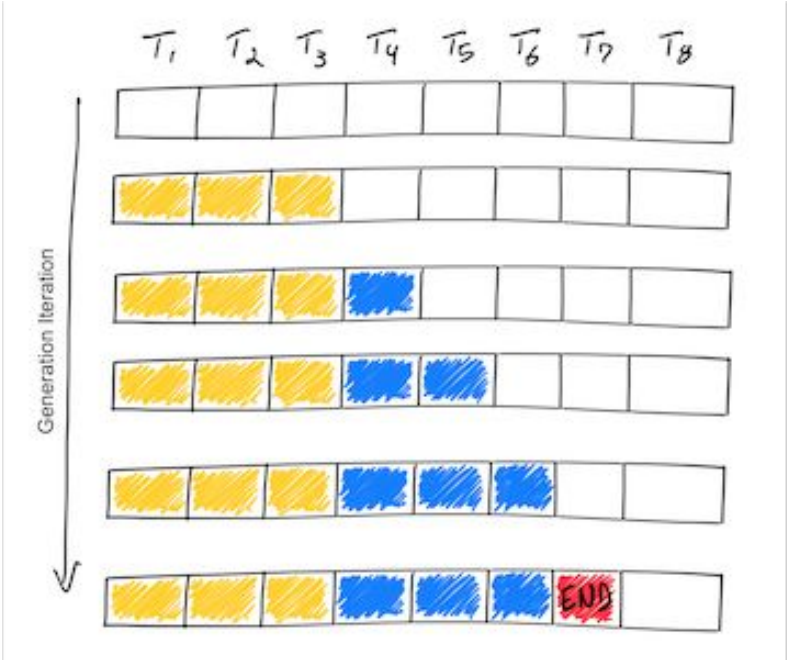
```
python finetune/adapter_v2.py \
    --data_dir data/alpaca  \
    --checkpoint_dir checkpoints/tiiuae/falcon-7b \
    --out_dir out/adapter/alpaca
```

У меня заняло ~**10 часов** для 52K инструкций



Inference

Batch Inference

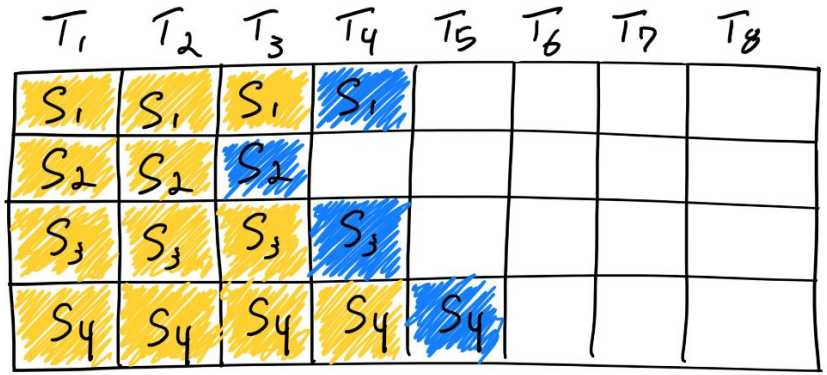


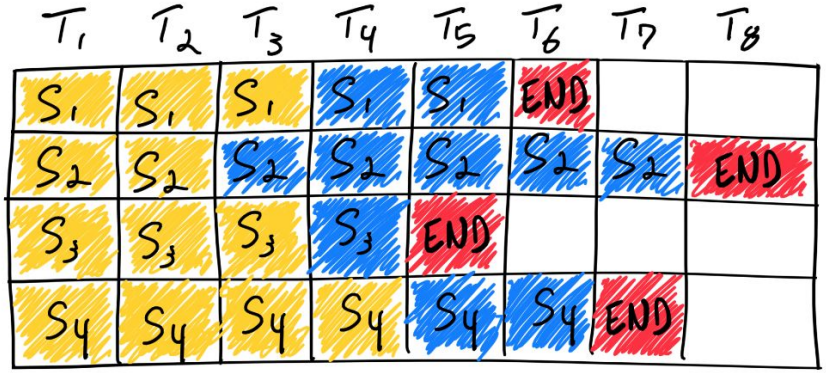
– prompt tokens

– generated tokens

– end-of-sequence token

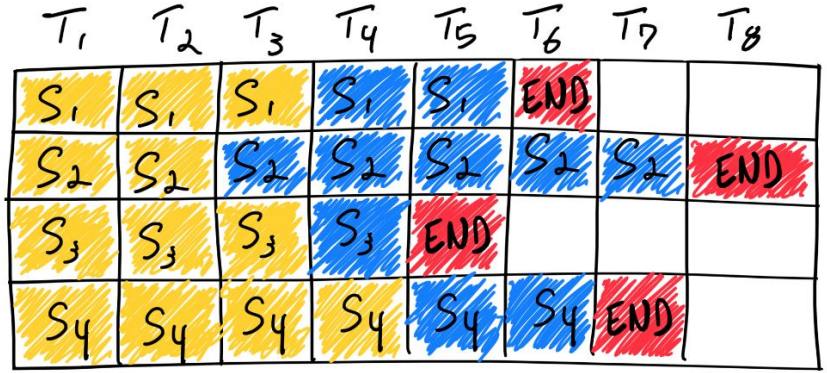
Batch Inference



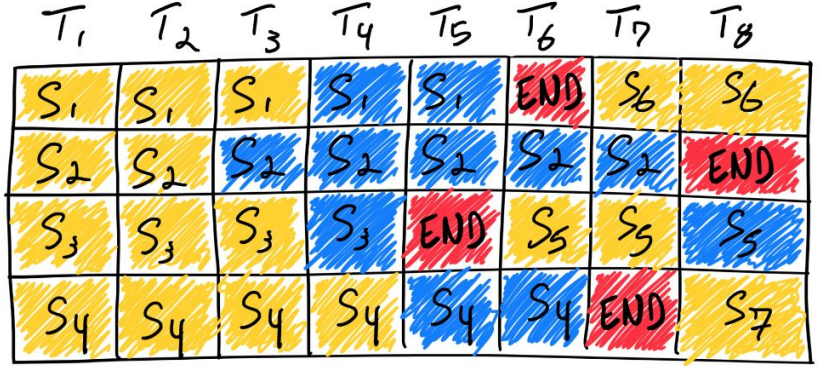


Naive batching

Batch Inference

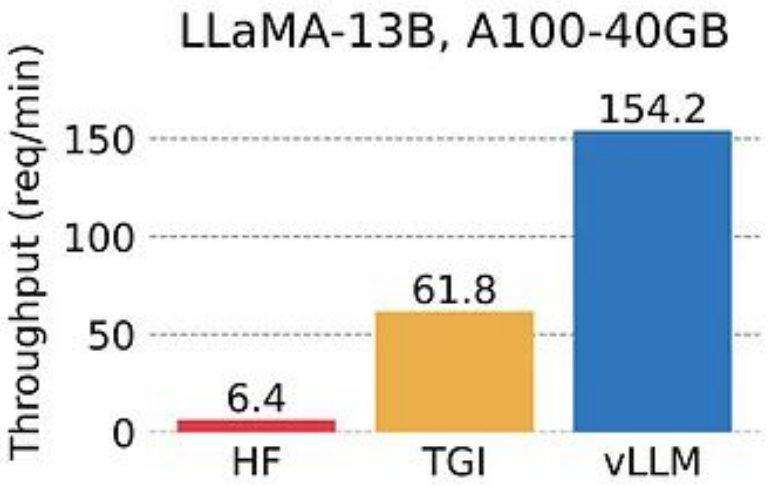


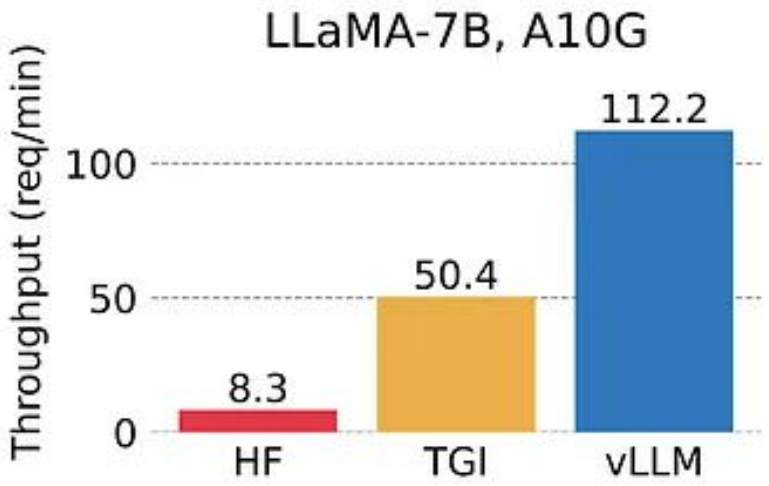
Naive batching



Continuous batching

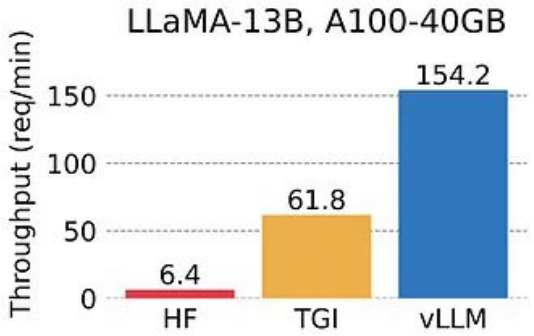
Batch Inference





Batch Inference

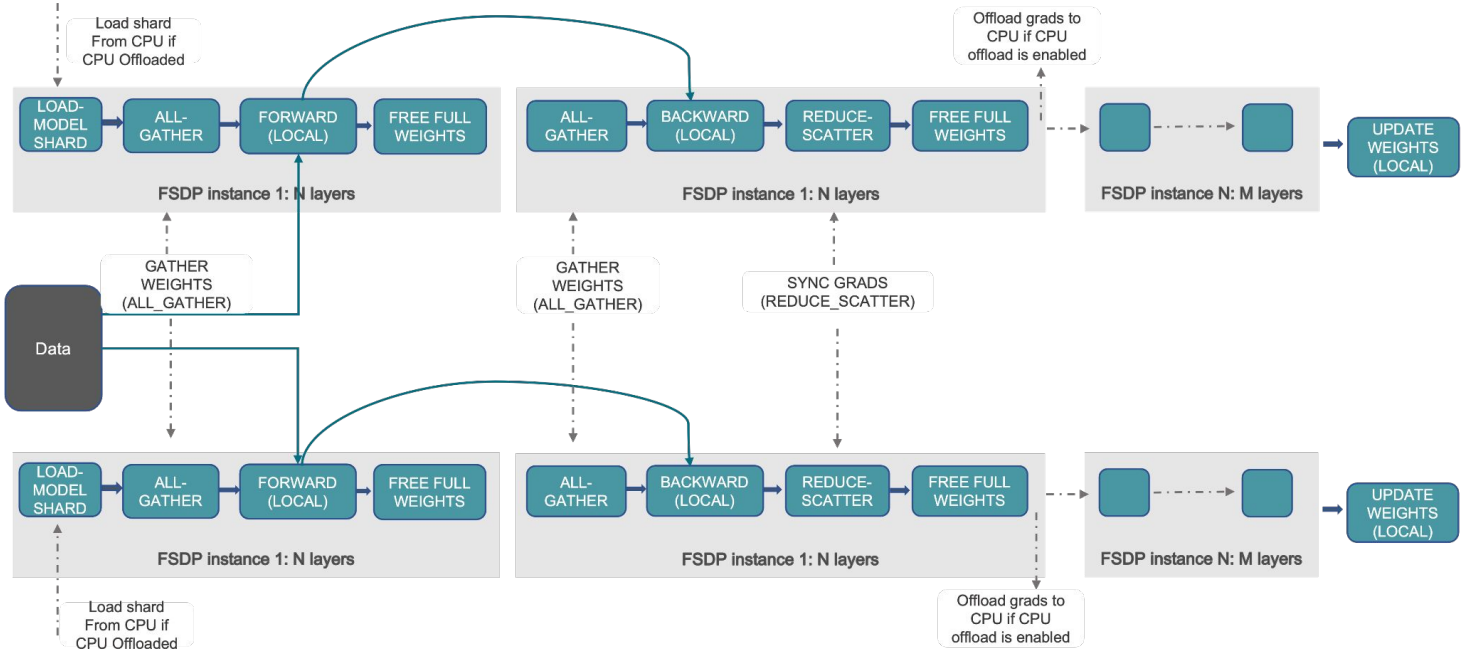
```python
from vllm import LLM, SamplingParams


llm = LLM(model="huggyllama/falcon-13b")
outputs = llm.generate(prompts, SamplingParams(temperature=0.8, top_p=0.95))

llm.generate(prompts, sampling_params)
```



vLLM - инференс сервер, который поддерживает разные трюки для ускорения LLMs

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

Multiple GPU devices



Fully Sharded Data Parallel (*FSDP) workflow*

Multiple GPU devices

```
python generate/base.py \
    --checkpoint_dir checkpoints/tiiuae/falcon-40b \
    --strategy fsdp \
    --devices 2 \
    --prompt "I am so fast that I can"
```

Falcon-7B

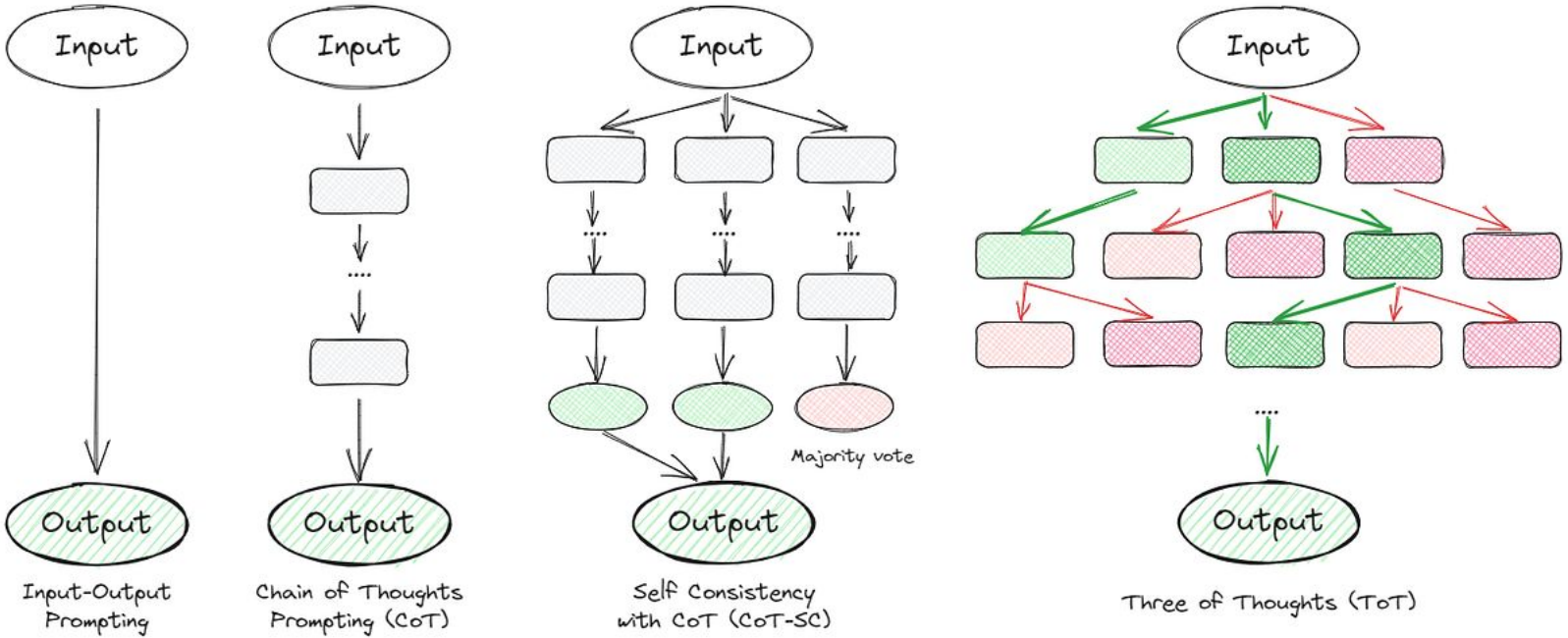
	1x A6000 (48 GB)	2x A6000 (48 GB)
Time for inference	1.65 sec	18.00 sec
Memory used	14.50 GB	9.60 GB

Falcon-40B

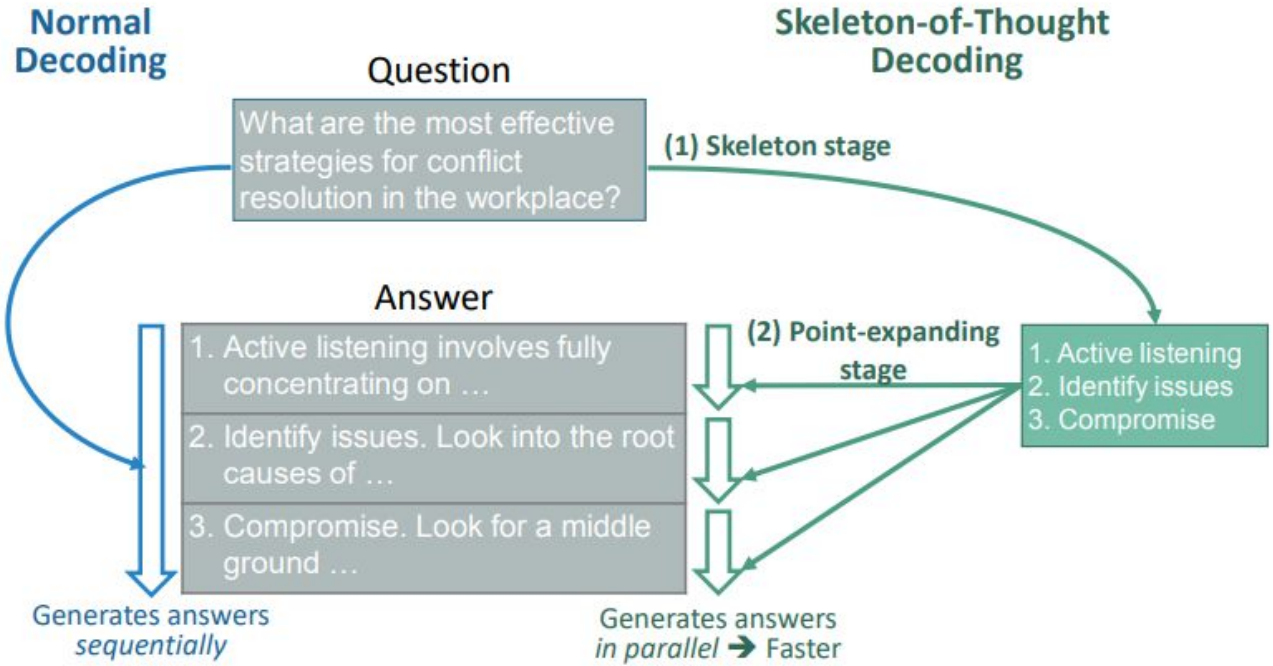
	1x A6000 (48 GB)	2x A6000 (48 GB)
Time for inference	- sec	83.40 sec
Memory used	- GB	46.10 GB

Prompt tuning

Chain of Thought



Skeleton-of-Thought



Замеры скорости и памяти

Falcon 7B & 1 A100 GPU

	Time for inference, tokens/sec	Memory used, GB
Vanilla Usage	33.9	28.9
Mixed-Precision	17.7	42.8
Bfloat16	42.4	42.8
Quantization	24.8	13.5
Continuous Batching	74.2	—
Multiple GPU	2.7	9.6

Фреймворки для деплоя

Framework	Tokens Per Second	Query per second	Latency	Adapters	Quantisation	Variable precision	Batching	Distributed Inference	Custom models	Token streaming	Prometheus metrics
vLLM	115	0.94	4.8 s	✗	✗	✗	✓	✓	✓	✓	✗
Text generation inference	50	0.26	4.8 s	✗	✓	✓	✓	✓	✓	✓	✓
CTranslate2	93	0.55	4.5 s	✗	✓	✓	✓	✓	✓	✓	✗
DeepSpeed-MII	80	0.47	2.5 s	✗	✗	✓	✓	✓	✗	✗	✗
OpenLLM	30	0.15	6.6 s	✓	✓	✓	✗	✗	✓	✗	✓
Ray Serve	28	0.15	7.1 s	✗	✓	✓	✓	✓	✓	✗	✓
MLC LLM	25	0.13	7.2 s	✗	✓	✗	✗	✗	✓	✓	✗

Comparison of frameworks for LLMs inference

Takeaways

1. Основные методы оптимизации такие же как и для DL моделей
2. Используйте одну модель и несколько адаптеров, если вам нужно решать сразу несколько доменных задач
3. Используйте готовые фреймворки для сёрвинга моделей – у них много фичей доступно сразу “из коробки”
4. По возможности используйте bfloat16 (нужен определённый тип GPU)
5. Изучите техники ускорения через разные стратегии промпт инжиниринга

Что читать дальше?

- Optimizing Memory Usage for Training LLMs and Vision Transformers in PyTorch
- Accelerating Large Language Models with Mixed-Precision Techniques
- Understanding Parameter-Efficient Finetuning of Large Language Models: From Prefix Tuning to LLaMA-Adapters
- The Secret Sauce behind 100K context window in LLMs: all tricks in one place
- 7 Frameworks for Serving LLMs
- 7 Ways to Speed Up Inference of Your Hosted LLMs

Спасибо за внимание!