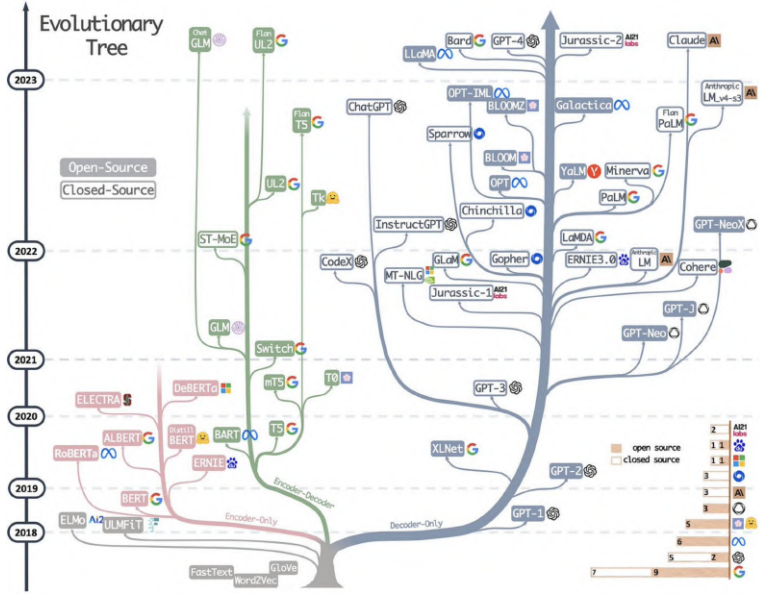
Как учить большие языковые модели

Май, 2024

Мурат Апишев
Search Tech Lead, Samokat.Tech
ex-Lead Data Scientist, SberDevices
mel-lain@yandex.ru

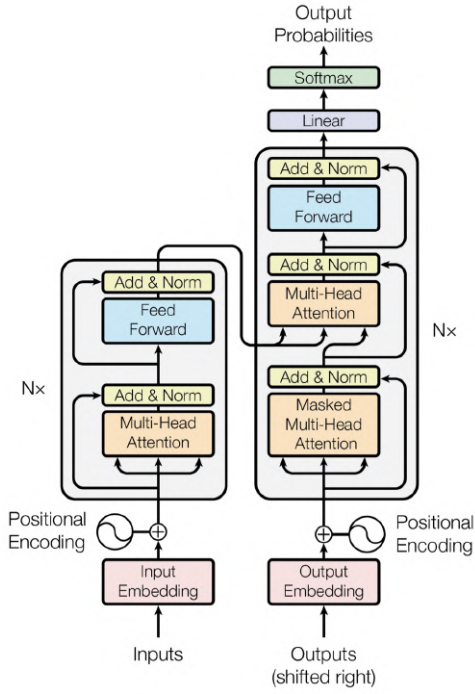
Large Language Model

- Основа современного AI
- Задача языкового моделирования на токенах разных модальностей
- Мультимодальные, мультидоменные, мультиязычные, инструктивные
- Сотни открытых и проприетарных моделей
- Доминирующая нейросетевая архитектура — Transformer



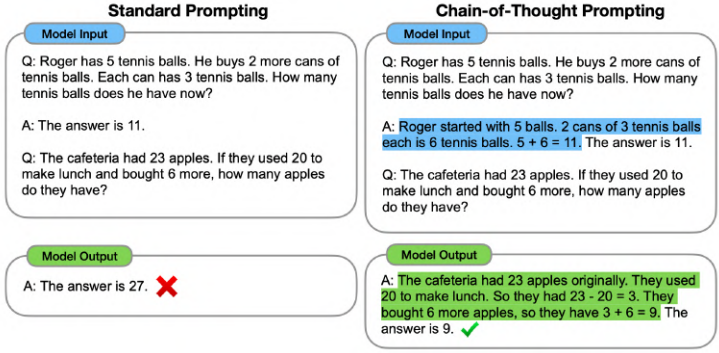
Transformer

- Multi-Head Attention и полносвязные слои
- Кодировщик-декодировщик (модели типа T5)
- Декодировщик часто используют отдельно (модели типа GPT)
- Обычно LLM обучаются на основе декодировщика Transformer (GPT, LLaMA, Qwen, Mistral, ...)
- Тексты перед обработкой токенизируются (BPE, Unigram, Wordpiece)



Промптинг LLM

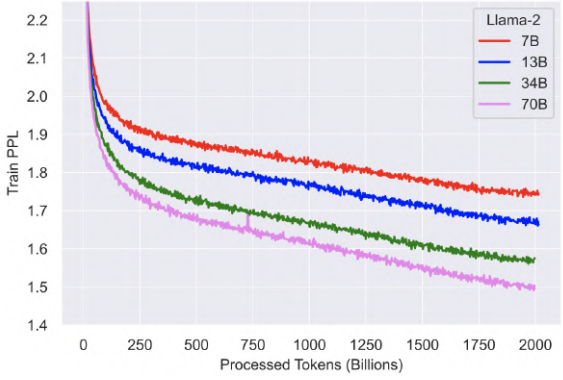
- Инструктивные LLM могут решать разные задачи, получая их на естественном языке (prompt), как исполнитель-человек
- Поставить задачу можно разными способами:
  - Zero-shot
  - Few-shot
  - Chain-of-Thought
  - Tree-of-Thought



- Авторы учат модель как можно более широкому пониманию инструкций, пользователи ищут лучшие варианты для своих задач
- Больше примеров в данных — больше возможностей

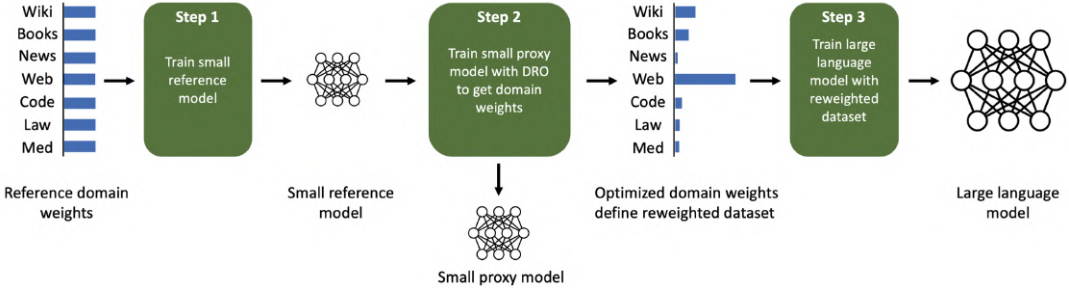
Этапы обучения: pre-train

- Первая стадия обучения LLM — предобучение (pre-train)
- Модель учится предсказывать следующий токен по контексту слева
- Если учить с Teacher Forcing — контекст берётся из обучения, если без — из того, что сгенерировала в процессе сама модель (комбинируют)
- На этом этапе приобретает основные знания о языке и мире



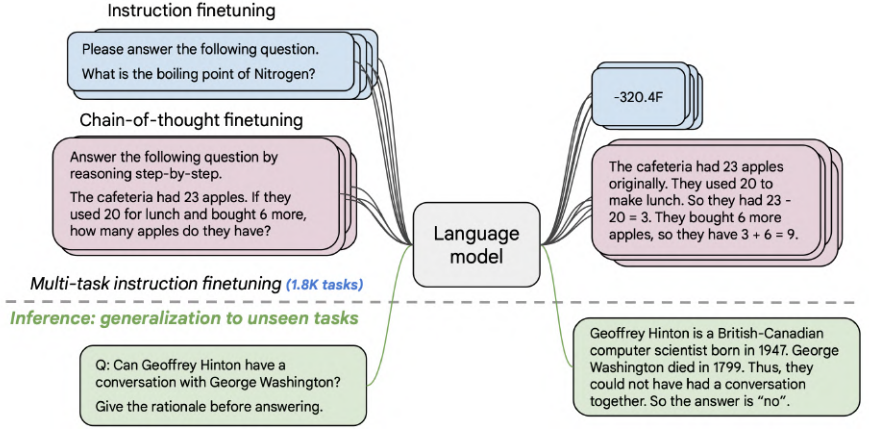
Данные для предобучения

- Для обучения сильной LLM требуются терабайты данных, это много
- Но всё равно меньше, чем данных существует в природе
- И не все данные одинаково полезны, какие-то вредны (их фильтруют), какие-то ничего не привносят, но тратят вычисления
- DoReMi — пример того, как путём грамотного перевзвешивания доменов сильно уменьшить объём данных и вычислений с улучшением качества модели на few-shot задачах:



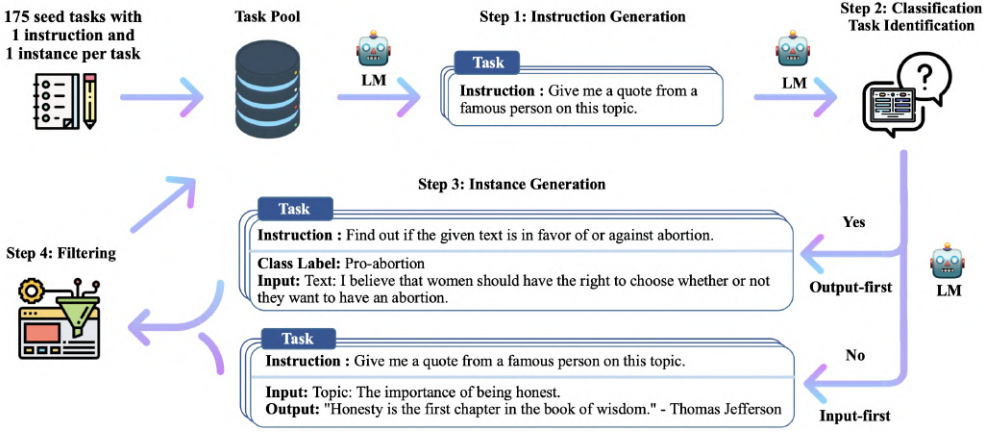
Этапы обучения: SFT

- Вторая стадия — Instruction Tuning (SFT)
- Модель учится понимать и исполнять запросы людей на естественном языке и вести диалоги
- Например: pre-train — LLaMA, instruct-tuned — Alpaca или Vicuna



Данные для SFT

- По сути модель так же обучается предсказывать следующий токен по прошлым, но на данных специального вида
- Объем данных сильно меньше, но требования к качеству высокие
- Сбор инструктивных датасетов сложен и дорог, пробуют Self-Instruct:



Данные для SFT

- Работа над данными и на претрейне, и на SFT позволяет сократить их объём и повысить качество, пример — кодовая модель phi-1 (и далее)
- Размеры относительно малы (до 1.3B), а качество сопоставимое с моделями в несколько раз больше
- Обучение на фильтрованных и синтетических данных, дообучение на качественных кодовых данных в стиле учебников

High educational value

```
import torch
import torch.nn.functional as F

def normalize(x, axis=-1):
    """Performs L2-Norm."""
    num = x
    denom = torch.norm(x, 2, axis, keepdim=True)
    .expand_as(x) + 1e-12
    return num / denom

def euclidean_dist(x, y):
    """Computes Euclidean distance."""
    m, n = x.size(0), y.size(0)
    xx = torch.pow(x, 2).sum(1, keepdim=True).
    expand(m, n)
    yy = torch.pow(x, 2).sum(1, keepdim=True).
    expand(m, m).t()
    dist = xx + yy - 2 * torch.matmul(x, y.t())
    dist = dist.clamp(min=1e-12).sqrt()
    return dist
```

Low educational value

```
import re
import typing
...

class Default(object):
    def __init__(self, vim: Nvim) -> None:
        self._vim = vim
        self._denite: typing.Optional[SyncParent]
    = None
        self._selected_candidates: typing.List[int
    ] = []
        self._candidates: Candidates = []
        self._cursor = 0
        self._entire_len = 0
        self._result: typing.List[typing.Any] = []
        self._context: UserContext = {}
        self._bufnr = -1
        self._winid = -1
        self._winrestcmd = ''
        self._initialized = False
```

Данные для SFT

- Почти все знания модель получает на этапе предобучения
- Идея LIMA:
  - на SFT не нужно вкладывать в модель новую информацию
  - нужно как можно лучше объяснить модели, как общаться
- Для этого данных должно быть немного, но очень высокого качества
- Собранный вручную набор из 1000 примеров для LLaMA 65B позволил обучить модель высокого качества
- Гипотеза: хорошую SFT-модель не нужно выравнивать (будет дальше)

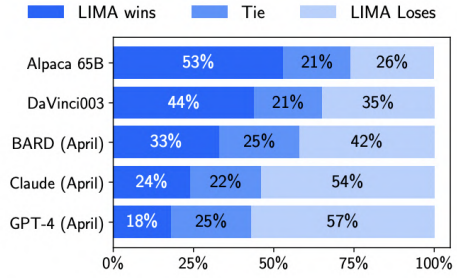


Figure 1: Human preference evaluation, comparing LIMA to 5 different baselines across 300 test prompts.

Данные для SFT

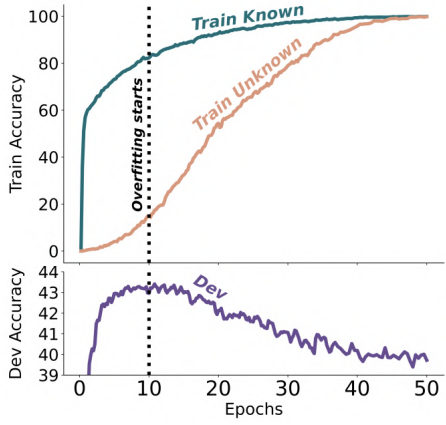
- Небольшие модели часто учат на выходах больших, и они часто вместо повторения рассуждений просто повторяют стиль
- В Orca предлагается дообучить LLaMA 13B на большом объёме специально собранных синтетических данных:
  - разнообразные задания и инструкции набираются из данных FLAN, 2021
  - модель-учитель получает их на вход с дополнительными инструкциями, требующими детального объяснения ответа, например:

    «You should describe the task and explain your answer. While answering a multiple choice question, first output the correct answer(s). Then explain why other answers are wrong. Think like you are answering to a five year old.»

  - модели генерируют ответы с объяснениями
- Модель-ученик тренируется на полученных тройках «системный промпт»-«задание»-«полный ответ учителя»
- Разнообразие и качество данных дополняется количеством: 1M сэмплов на основе GPT-4 и 5M — на основе ChatGPT

Данные для SFT

- Фактологическая информация на SFT не должна противоречить данным в pre-train
- Иначе модель начинает сильнее галлюцинировать
- Модель должна получать данные, в которых она либо сильно уверена, либо почти уверена (без этой категории качество ниже)
- В работе предлагается метод проверки корректности факта путём многократной проверки модели с few-shot промптом
- Модель выучивает новую информацию медленее и с потерей качества

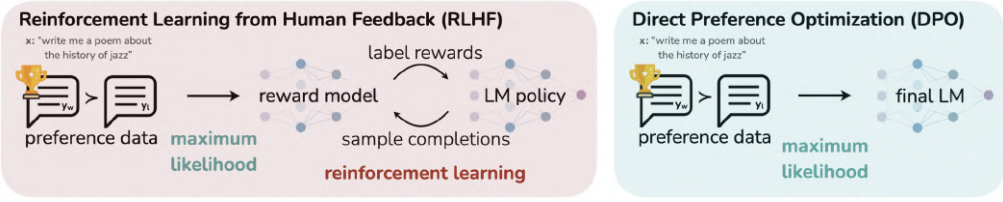


Этапы обучения: alignment

- Третий, опциональный, шаг — выравнивание (alignment)
- Диалоговая модель дообучается для генерации более корректных, полезных и безопасных ответов
- Популярная техника, использованная в Instruct GPT — RLHF:
  - обученная LLM генерирует на тестовом наборе инструкций ответы
  - ответы размечаются асессорами, на их ответах учится сильная reward-модель, она оценивает по тексту его качество
  - заводится две копии модели (A) и (B), учится (A)
  - обе модели генерируют ответы на каждый промпт, ответ (A) оценивается reward-моделью
  - веса (A) обновляются так, чтобы максимизировать reward и не давать ответы, очень далёкие от исходной (B)
  - расстояние определяется по KL-дивергенции между выходными распределениями моделей
  - обновление весов идёт по заданному алгоритме (например, PPO)

Этапы обучения: alignment

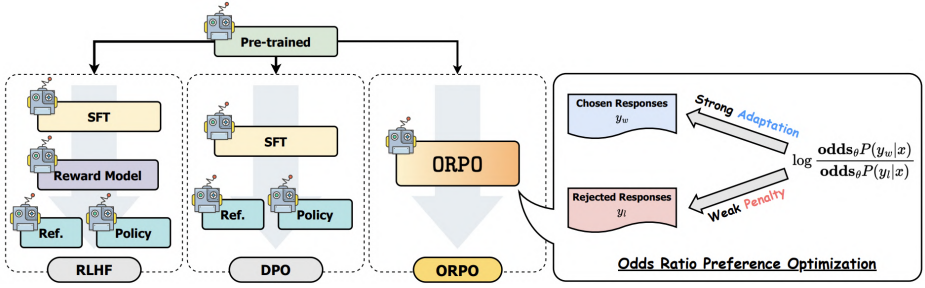
- RLHF сложен и не всегда работает хорошо, альтернатива — DPO:



- DPO устраняет необходимость обучения отдельной reward-модели и онлайн-генерации с RL
- Вместо этого функция потерь LM переопределяется и оптимизируется напрямую
- Она учитывает как предпочтения ответов из набора данных, так и требование не уводить ответы далеко от исходной модели
- DPO использует только бинарные оценки предпочтений, иные форматы нужно сводить к этому

Этапы обучения: alignment

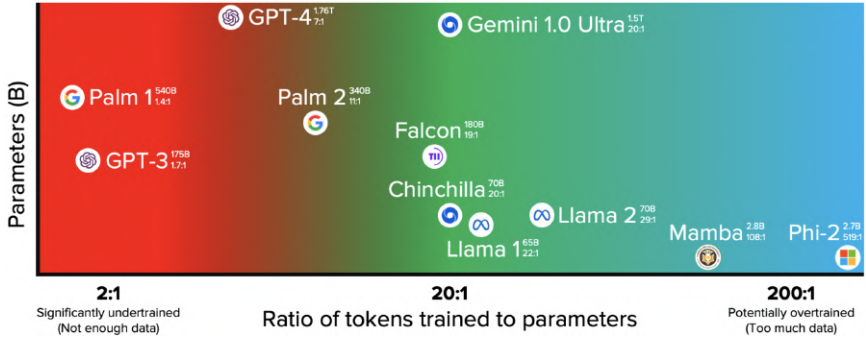
- Правильно приготовленный SFT может сам выровнять модель:



- SFT поощряет позитивные примеры, но не штрафует дополнительно за негативные: модель выучивает домен, а не предпочтения
- **Решение**: добавить в loss слагаемое, повышающее вероятность генерации последовательности токенов позитивного ответа по сравнению с негативным
- Метод ORPO хорошо показал себя при SFT Phi-2, LLaMA 2 и Mistral

Баланс между параметрами и данными

- Авторы Chinchilla, 2022 задумались о важности баланса между размерами модели и объёмом данных (длительностью обучения)
- На обучение модели выделяется ограниченный вычислительный бюджет
- Его можно тратить, увеличивая либо размер модели, либо длительность её обучения, нужно балансировать, максимизируя loss
- Большинство моделей оказались обученными неоптимально:

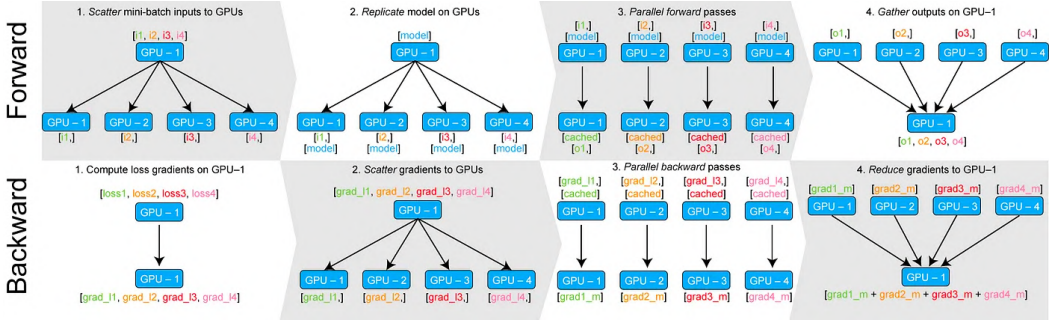


Масштабирование обучения

- При обучении память GPU в основном тратится на
  - веса модели
  - состояние оптимизатора
  - активации
  - градиенты
- Объем памяти GPU сильно ограничен: A100 имеет 80Гб
- У современных моделей даже веса могут не влезать в такой объём
- При обучении с помощью стандартного AdamW состояние оптимизатора требует x2 от размера модели
- Для обучения больших моделей с адекватной скоростью требуется обрабатывать большой объём данных одновременно
- Это возможно только при использовании параллельных вычислений на множестве GPU, этот процесс можно организовать по-разному

Data Parallelism

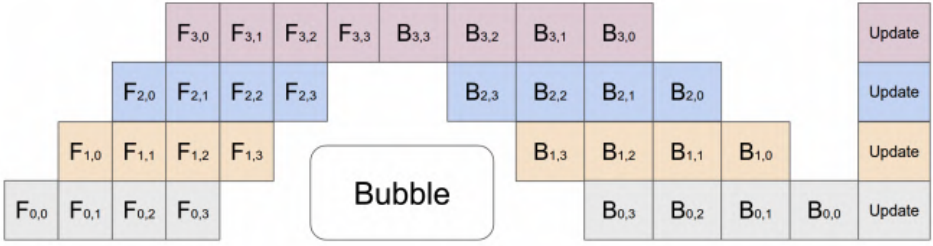
- Если модель и состояние оптимизатора не занимают всю память GPU, то очевидный способ параллелизма — по данным (Data Parallelism)
- Каждая GPU имеет свою копию модели и обрабатывает часть батча



- Можно добавить Gradient Accumulation: разделять батч по всем GPU не целиком, а частями для экономии памяти
- Градиенты агрегируются до обработки всего батча, после чего запускается обновление параметров модели

Pipeline Parallelism

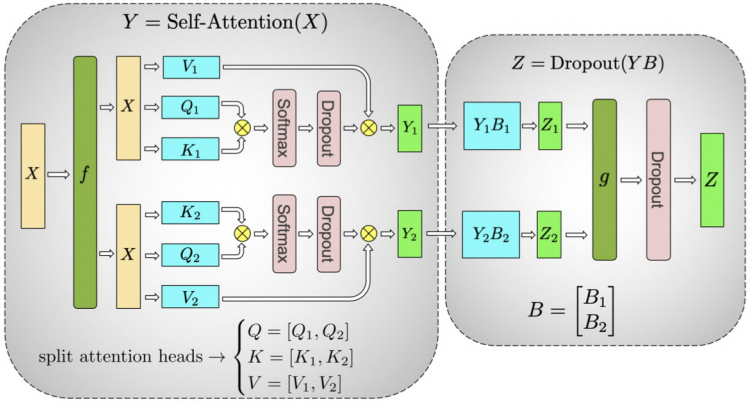
- Если памяти одной GPU не хватает, модель можно разрезать и разложить на несколько устройств
- Pipeline Parallelism — группы слоёв раскладываются по своим GPU
- Для уменьшения простоя батч нарезается на части, и более глубокие слои начинают работать раньше



- По сравнению с другими подходами требует сильно большего переписывания кода

Tensor Parallelism

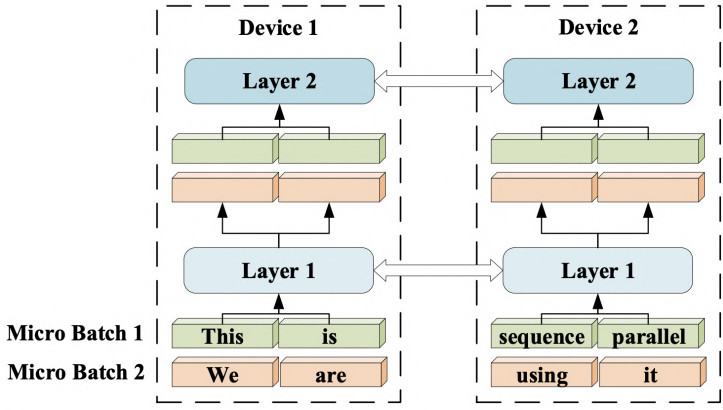
- Tensor Parallelism: по GPU раскладываются части тензора



- Self-attention параллелится естественно за счёт разных голов
- При TP сетевые коммуникации более интенсивные, чем при DP или PP
  $\Rightarrow$ модель лучше раскладывать на одном узле DGX или в сети InfiniBand

Sequence Parallelism

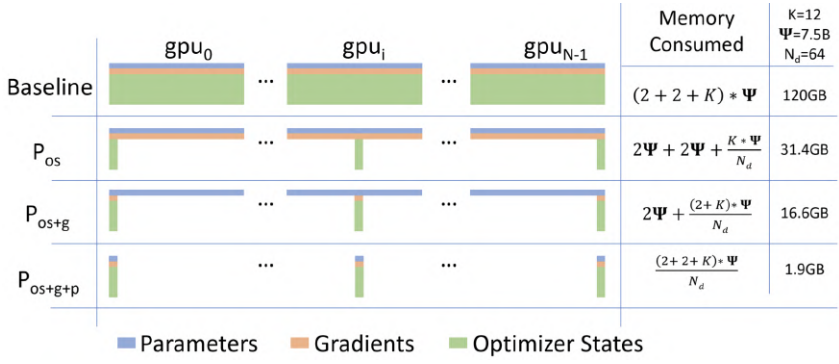
- Sequence Parallelism: по GPU раскладываются части последовательности



- Может естественно комбинироваться с Ring Attention (будет дальше)
- В определённых случаях эффективнее (за счёт меньшего объёма коммуникаций), чем TP, но может со всем комбинироваться

Zero Redundancy Optimizer

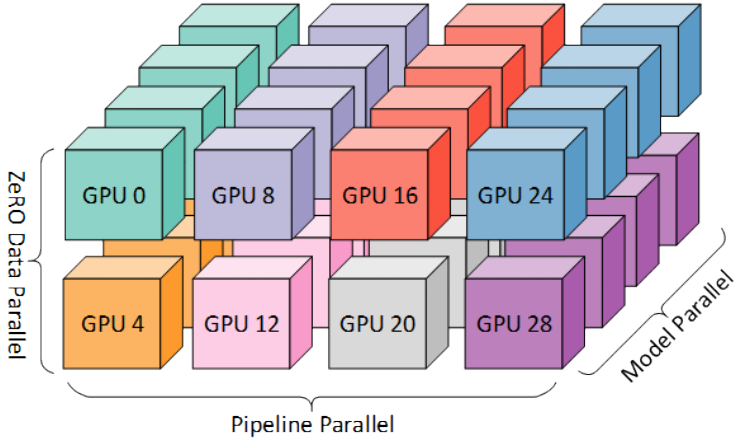
- В Data Parallelism можно избежать хранения избыточной информации с помощью ZeRO, 2019, стандартная реализация — DeepSpeed
- На каждом этапе (stage) потребление падает, сетевые коммуникации растут x1.5 только на этапе 3



- ZeRO умеет выгружать данные в RAM, что тоже экономит память GPU
- Более современный конкурент со схожим функционалом — FSDP, 2022

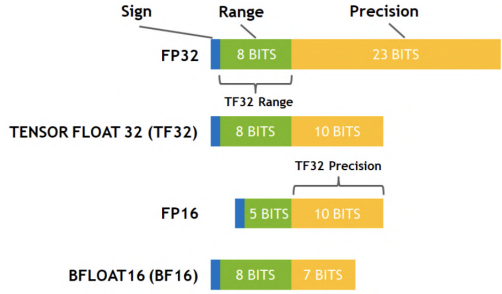
3D-параллелизм

- Все техники могут применять как сами по себе, так и в комбинациях
- DP+PP+TP дают 3D-параллелизм, часто комбинируется с ZeRO stage 1 (stage 2/3 тоже можно, но сложнее + растут сетевые коммуникации)



Эффективность обучения

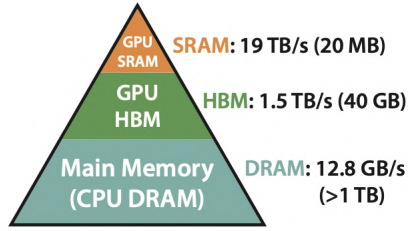
- Обучать модели в fp32 неэффективно
- Используют Mixed Precision:
  - две копии весов, в fp32 и fp16 / bf16 (если GPU Ampere)
  - активации считаются в fp16 / bf16
  - агрегации и нормализации в fp32
  - градиенты и состояние оптимизатора в fp32



- Затраты памяти нивелируются большим батчем, а обучение ускоряется
- fp16 требует масштабирования loss для стабильности (умножение на коэффициент и обратно), bf16 — нет, он в целом более стабилен
- GPU Ampere могут заменять fp32 на tf32 — более эффективный и экономичный формат, можно комбинировать это с Mixed Precision

Flash Attention

- Утилизация GPU была очень неоптимальной — много времени уходило не на вычисления, а на пересылку данных между HBM и SRAM
- Подсчёт softmax в self-attention генерирует промежуточные матрицы, которые занимают место и перемещаются

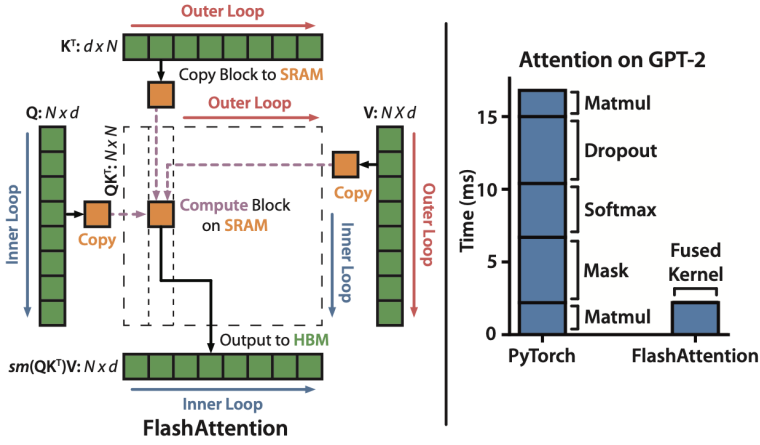


Memory Hierarchy with Bandwidth & Memory Size

- Можно ввести дополнительные переменные и считать softmax блочно
- Не нужно хранить промежуточные матрицы, передача данных между HBM и SRAM становится экономичнее
- Нужные для backward-шага промежуточные значения можно эффективно пересчитывать вместо хранения на forward

Flash Attention

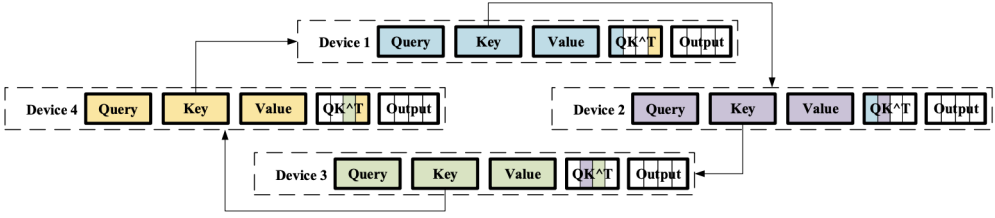
- Дополнительное ускорение получено за счёт Fusing — выполнения набора операций одним CUDA ядром



- Flash Attention 2, 2023 даёт ещё больший выигрыш по скорости за счёт вычислительных оптимизаций на GPU

Ring Attention

- Метод масштабирования точного подсчёта внимания на множество GPU для увеличения длины последовательности — Ring Attention
- Запросы Q, ключи K и значения V разбиваются на блоки по числу GPU и распределяются по ним
- Вычисления self-attention блочное, на каждой GPU:
  - для блока Q вычисляются все коэффициенты с текущим блоком K
  - блок K передаётся на следующую GPU, а текущая получает новый блок с предыдущей карты, и вычисление повторяется
  - после полного круга подсчёт идёт для V и агрегируется полный attention

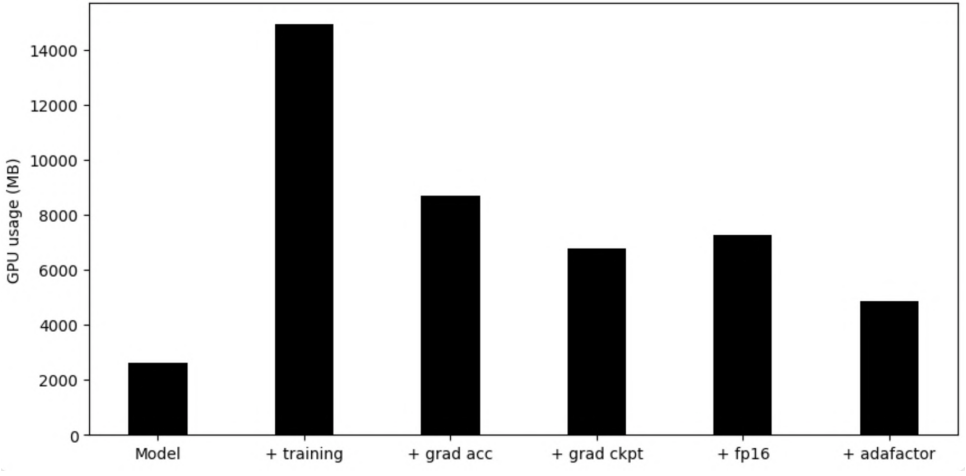


Gradient Checkpointing

- Важная техника для оптимизации памяти на полносвязных слоях — Gradient (Activation) Checkpointing
- Выходы каждого линейного слоя на forward-шаге нужны для вычисления градиентов на этом слое на обратном шаге, поэтому они сохраняются
- Это приводит к линейному по числу слоёв росту потребления памяти
- Можно ничего не хранить и вычислять для каждого слоя активации с нуля (т.е. от начала сети до этого слоя)
- Это экономит память, но объем вычислений на forward из линейного по числу слоёв становится квадратичным
- **Решение:** сохранять активации части слоёв на некотором расстоянии друг от друга (checkpoint)
- Вычисление активаций слоя будет идти от последнего чекпойнта
- В среднем потребление памяти падает с $O(n)$ до $O(\log n)$ за счёт замедления примерно на 20%

Комбинирование методов

- Можно экспериментировать с оптимизаторами, например, Adafactor более экономичный по памяти, чем AdamW
- Большинство оптимизаций хорошо комбинируются друг с другом:



https://huggingface.co/docs/transformers/v4.20.1/en/perf_train_gpu_one

Алгоритмы оптимизации

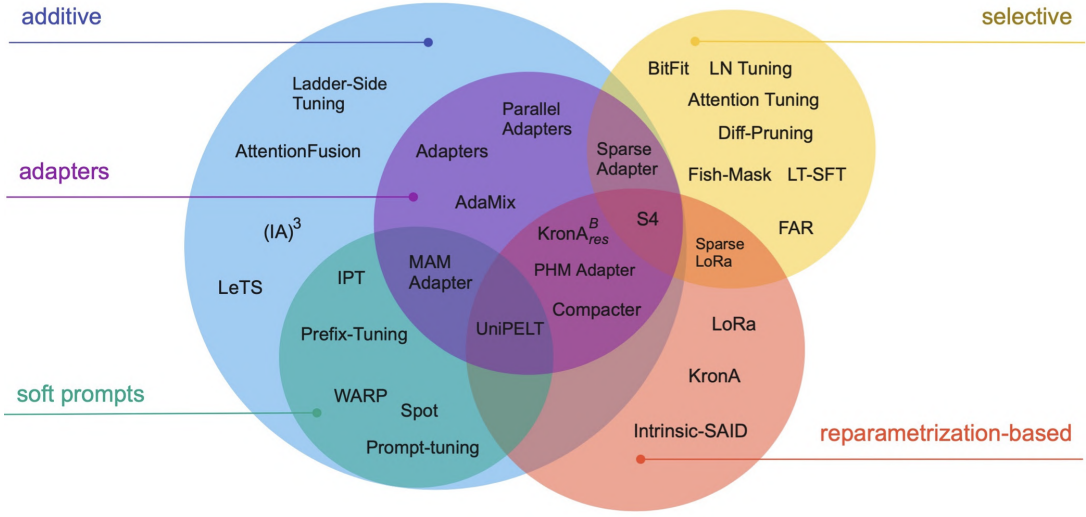
- Стандартный алгоритм обучения LLM — AdamW, 2017:
  - в основе лежит Adam, 2014 (Momentum, 1986 + RMSprop, 2012)
  - моменты 1-го и 2-го порядков считаются, хранятся и используются на шаге обновления весов
  - в отличие от Adam, AdamW делает weight decay регуляризацию на параметрах, а не на градиентах (работает лучше, чем L2 на loss)
- Популярные альтернативы:
  - Adafactor, 2018
  - Shampoo, 2018
  - Adan, 2018
  - AMSGrad, 2019
  - Sophia, 2023
  - Lion, 2023
- Полноценно обойти AdamW сложно — для прочих алгоритмов слишком мало разносторонних экспериментов, проводить их долго и дорого
- Для уменьшения потребления памяти используются квантизованные варианты алгоримов (например, 8-bit AdamW, 2021)

Эффективность дообучения

- Стандартный подход в использовании LLM — Transfer Learning
- Большая и умная модель адаптируется под частные задачи с помощью дообучения на небольшом наборе данных
- **Проблема:** дообучение LLM целиком может требовать больших ресурсов и времени
- **Возможное решение:** учить не всю модель, а только отдельные слои
- **Проблема:** задачи могут быть многочисленными и разнообразными — не хочется на каждую учить, хранить и хостить целую модель
- Альтернатива полному или частичному дообучению — адаптеры
- Модель остаётся неизменной, к ней как-то добавляются немного новых параметров (или используется малая часть исходных весов)
- При дообучении эти параметры настраиваются корректировать работу модели для повышения качества на целевой задаче

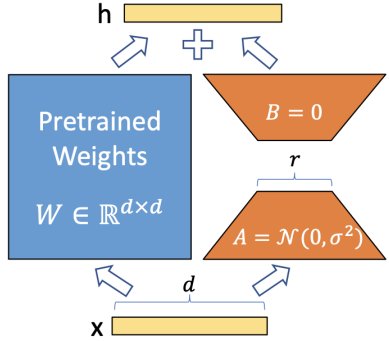
Зоопарк адаптеров

- Адаптеров придумали очень много:



Low-Rank Adaptation

- Одним из наиболее популярных методов остаётся LoRA:
  - веса модели полностью замораживаются, выбираются целевые линейные веса
  - для каждой матрицы весов заводится пара новых матриц — её низкоранговое разложение
  - при работе эти матрицы перемножаются и результат складывает с основными замороженными весами
  - хороший рецепт: добавлять адаптер на все матрицы весов запросов и значений в self-attention



Качество генерации

- Техника на основе идей ансамблирования: Checkpoint Averaging — усреднение весов нескольких версий модели в конце обучения
- Выбор и настройка **метода декодирования** выходов LLM:
  - Greedy search
  - Beam search (`num_beams`)
  - Top-K sampling (`top_k`)
  - Top-P [nucleus] sampling (`top_p`)
  - Contrastive search (`top_k, penalty_alpha`)
- Неочевидная проблема — выбор токенов на границе промпта и ответа:

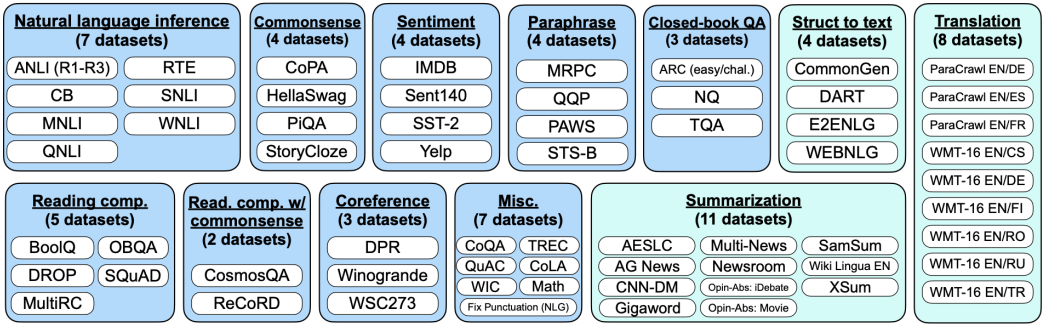
```
link is <a href="http: -> link is <a href="http: //site.com
link is <a href="http -> link is <a href="http://site.com
```

- Token Healing: до генерации откатиться на один или более токенов назад
- С любым промптом можно использовать Self-Consistency:
  - ответ модели генерируется несколько раз
  - результат — самый частый вариант (+ степень уверенности)

Checkpoint ensembles: Ensemble methods from a single training process, 2017
Contrastive Search Is What You Need For Neural Text Generation, 2022
https://github.com/guidance-ai/guidance

Оценка качества LLM

- Способности моделей проверяются путём решения разных текстовых или мультимодальных задач на разных наборах данных
- Схема различных NLP-задач и соответствующих данных (синие — с короткими ответами, бирюзовые — с длинными):



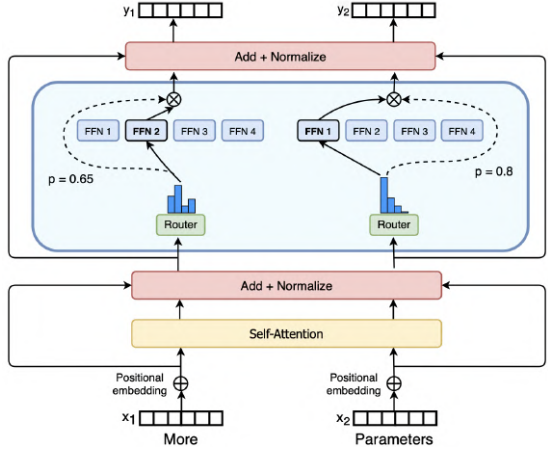
Оценка качества LLM

- Из наборов данных формируют бенчмарки, их много для разных задач, длин контекста, доменов, языков и модальностей, примеры:
  - MMLU, 2020 (тексты, английский)
  - HumanEval, 2021 (программный код, английский)
  - MMBench, 2023 (тексты и изображения, английский)
  - LongBench, 2023 (длинные тексты, английский)
  - MERA, 2023 (тексты, русский)
- Проверка коротких ответов автоматическая, с длинными сложнее — автометрики слабые, проверяют люди или более сильные LLM (GPT-4):

  «Ты выступаешь в роли асессора. Тебе покажут правильный пересказ текста и пересказ, сгенерированный моделью, твоя задача оценить по шкале от 1 до 10 качество генерации пересказа . . . »
- Качество оценки выше, если уточнить критерии оценки в промпте и попросить модель обосновать своё решение

За кадром: модификации архитектуры

- Замена нормировок и активаций: pre-LayerNorm, RMSNorm, SwiGLU
- Замена MHA: RWKV, RetNet, Mamba
- Замена FF: KAN
- Добавление экспертов и рутинга: MoE, DeepSeekMoE



LLaMA: Open and Efficient Foundation Language Models, 2023
RWKV: Reinventing RNNs for the Transformer Era, 2023
Retentive Network: A Successor to Transformer for Large Language Models, 2023
Mamba: Linear-Time Sequence Modeling with Selective State Spaces, 2023
KAN: Kolmogorov-Arnold Networks, 2024
Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity, 2021
DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models, 2024

За кадром: длинный контекст

Для обхода квадратичной сложности self-attention можно

- понижать сложность вычислений за счёт понижения размерностей матриц: Linformer (2020), Performer (2020)
- вычислять внимание по частям последовательности: Sparse Transformers (2019), Longformer (2020), LongNet (2023)
- обрабатывать последовательность иерархически: Long Document Summarization with Top-down and Bottom-up Inference (2022)
- вместо увеличения длины входа передавать контекст рекурентно: Transformer-XL (2019), RMT (2023)
- вместо увеличения длины входа сохранять контекст в kNN-индексе: Memorizing Transformers (2022), Unlimiformer (2023), Focused Transformer (2023)
- заменить внимание на другой механизм, схожий по качеству и скорости обучения, и вычисляемый на инференесе рекурентно (варианты SSM): RWKV (2023), RetNet (2023), Mamba (2023)

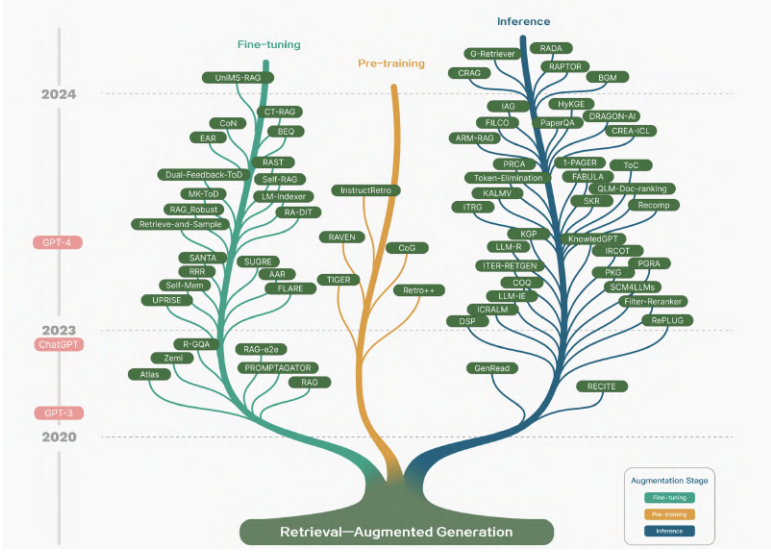
За кадром: позицинное кодирование

- Абсолютное кодирование сопоставляет позиции токена фиксированный синусоидальный или обучаемый вектор — не лучшее решение
- Относительные методы (кодируется расстояние между парой токенов) условно можно разделить на три вида:
  - репараметризация формулы подсчёта внимания
    - Transformer-XL, 2019
    - DeBERTa, 2021
  - обычная формула с добавлением обучаемого сдвига
    - T5, 2020
    - AliBi, 2022
  - ротационное кодирование (RoPE, 2021) и его вариации
    - xPos, 2023
    - Positional Interpolation RoPE, 2023
    - YaRN, 2023
- Пробовали и вообще обходиться без кодирования позиций в декодерах (NoPE, 2023), но работает не очень

За кадром: RAG и внешние вызовы

LLM не обязана содержать все знания в своих весах и уметь выполнять все задачи — есть Retrieval-Augmented Generation и вызовы внешних API

- LaMDA, 2022
- Gorilla, 2023
- Calcformer, 2023
- ToolLLM, 2023
- Toolformer, 2023
- FUXI, 2024



Спасибо за внимание!

Мурат Апишев
Search Tech Lead, Samokat.Tech
ex-Lead Data Scientist, SberDevices
mel-lain@yandex.ru