

# LLM: fine-tuning

Natural Language Processing

Ника Зыкова, 2025/11/27

# Дообучение: что это

Дообучение - адаптация LLM к конкретным задачам/доменам/требованиям.

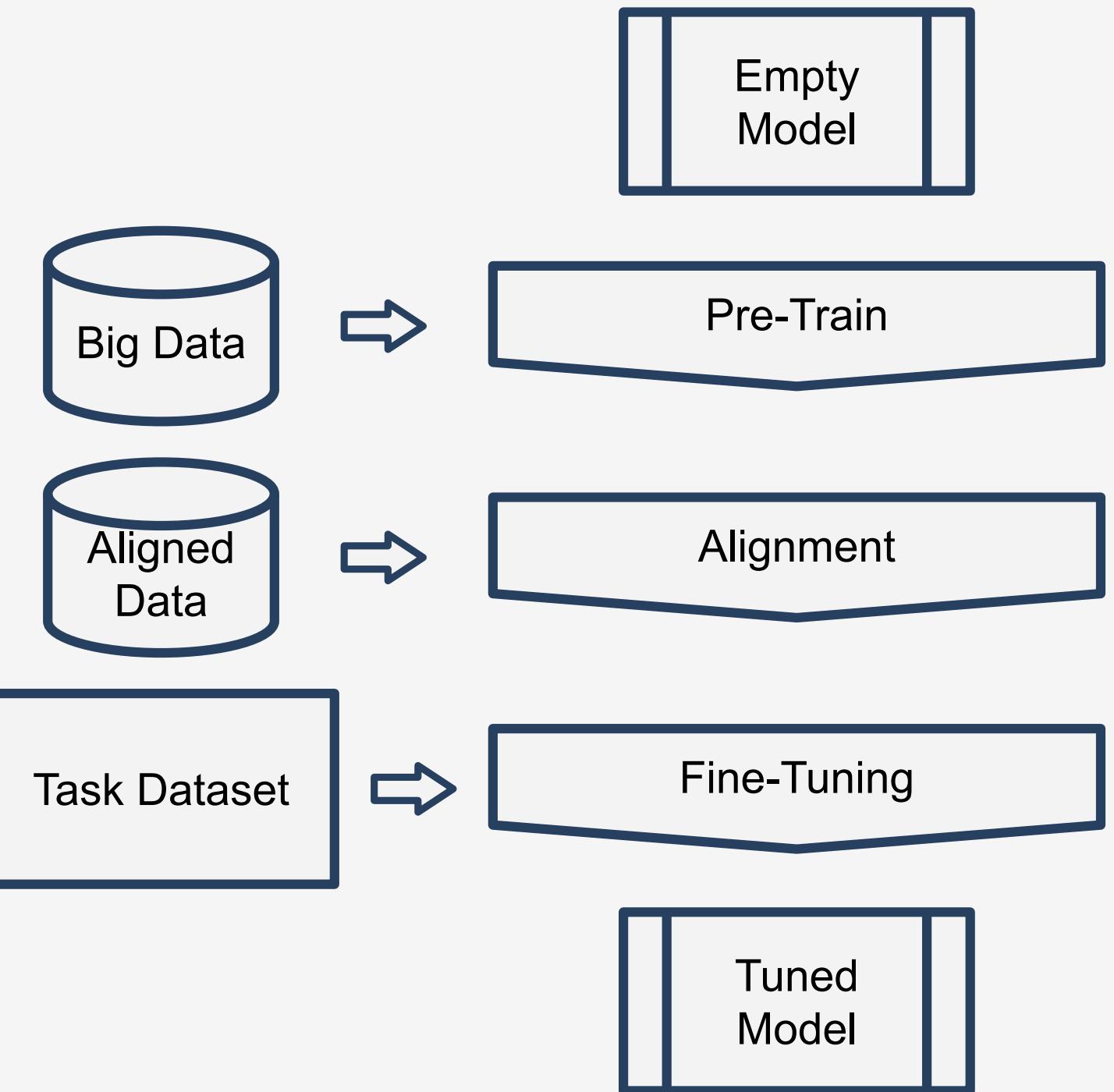
Отличие от претрэйна:

- Меньшая вычислительная стоимость: меньше данных, иногда меньше обучаемых параметров;
- Размеченные данные под конкретную задачу.

Отличие от alignment:

- Более конкретная задача / более узкий домен;
- Чаще всего более короткое обучение.

Но граница между alignment и fine-tuning сильно более тонкая.

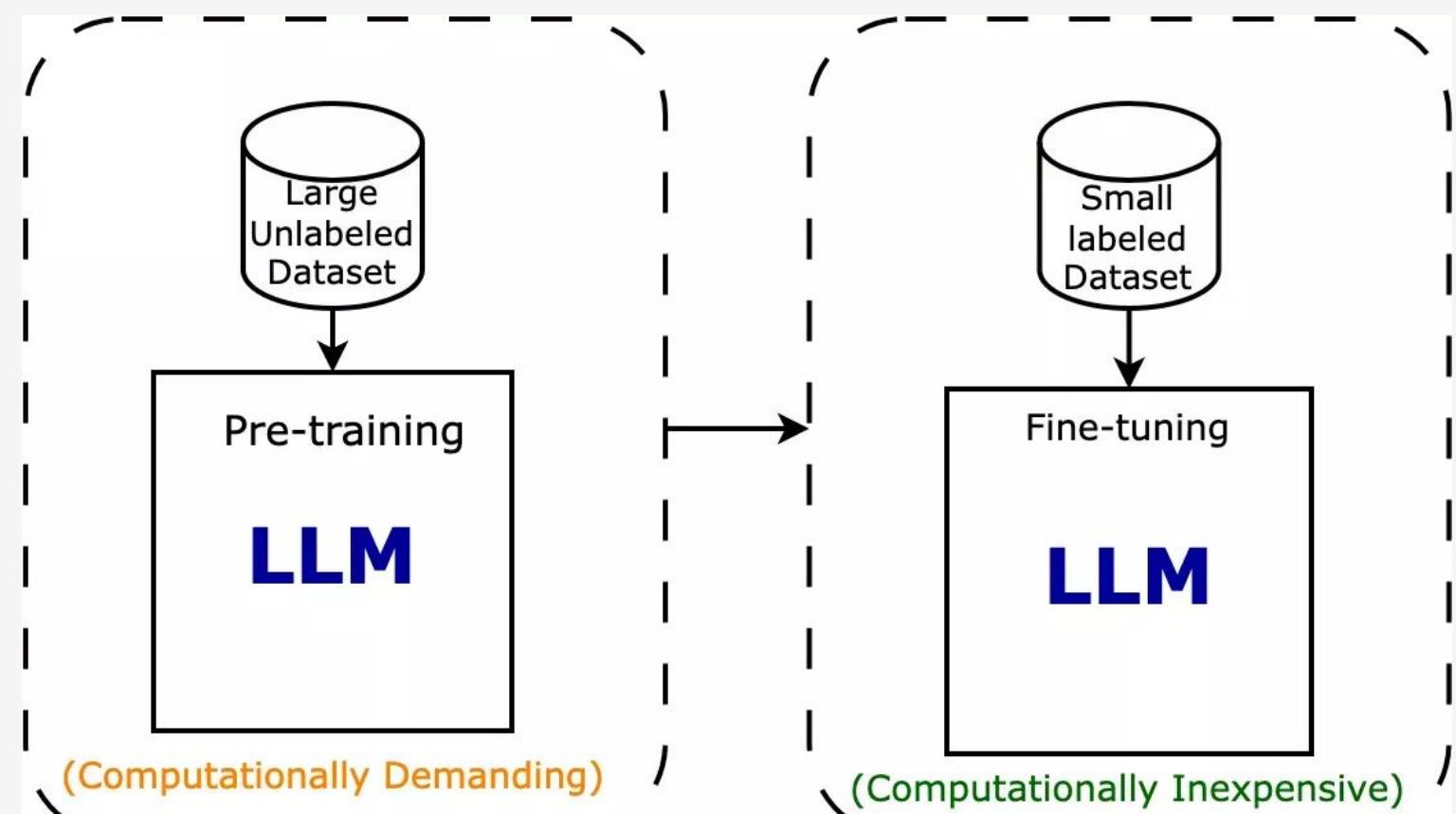


# Дообучение: зачем

Основная цель дообучения - улучшить качество на одной конкретной задаче (часто в ущерб остальным).

Когда это оправдано и возможно:

- У вас есть достаточное кол-во размеченных данных и вычислительных ресурсов;
- Промптинг показал себя недостаточно хорошим и слишком дорогим / медленным;
- Модель необходимо встроить в пайплайн, из-за чего нужно работать с логитами;
- Особеностей задачи слишком много и модель не знает их "из коробки".



# В чем разница

## Промптинг

- Не изменяет модель;
- Не требует датасета для создания решения;
- Низкая способность к адаптации (модели бывают упрямыми);
- Большой расход ресурсов на инференсе: промпт часто длинный;
- Хорош для создания прототипа и проверки гипотез.

## Дообучение

- Изменяет параметры модели (частично или полностью);
- Требует данных для обучения и проверки;
- Хорошая адаптация и стабильное поведение;
- Требует вычислительных ресурсов на обучение;
- + 1 модель в проде, что тоже требует ресурсов.

## Retrieval Augmented Generation

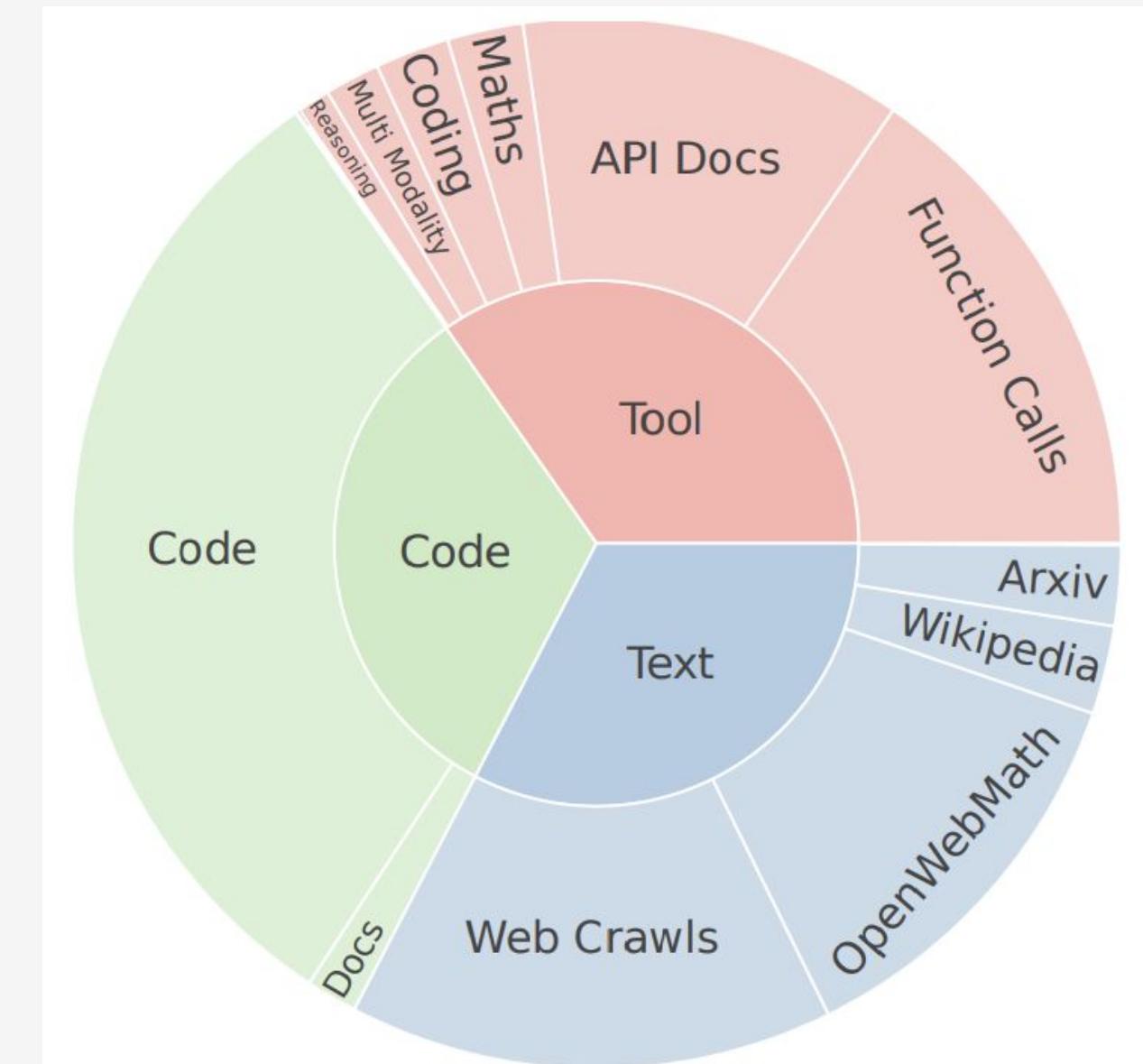
Даем модели релевантный кусок из базы знаний при генерации, чтобы она могла применять внешнюю информацию.

- В простом варианте не требует дообучения, но требует базу знаний;
- Полезно, когда модель должна владеть большим кол-вом фактов.

# Требования к данным

Чтобы дообучение было успешным, необходимо соблюдать несколько требований к данным:

- ❖ Качество > количество: модель изначально многое знает, на этапе дообучения грязные данные могут ее сломать (но часто есть пороговое кол-во, ниже которого нельзя);
- ❖ Разметка в едином формате, соответствующем будущему инференсу;
- ❖ Данные должны быть разнообразные, сбалансированные и охватывать различные краевые случаи;
- ❖ Негативные примеры, если технически возможно их использовать.



(a) Hephaestus-Forge

# SFT

SFT (Supervised Fine-Tuning) - классическое обучение с учителем, от претрэйна отличается тем, что мы учимся только на выходе модели после промпта.

Плюсы:

- Сильное влияние на поведение;
- Прямая зависимость между обучением и целевым качеством (при хороших данных).

Минусы:

- Много вычислительных ресурсов;
- Все проблемы обучения: забывание, нестабильность, переобучение.

## Instruction

Проанализируй список расходов и вычисли общую сумму:

Хлеб: 55

Молоко: 80

Фрукты: 230

Ответ верни в виде: "Итого: X рублей".

## Output

Итого: 365 рублей.

# Частичное обучение

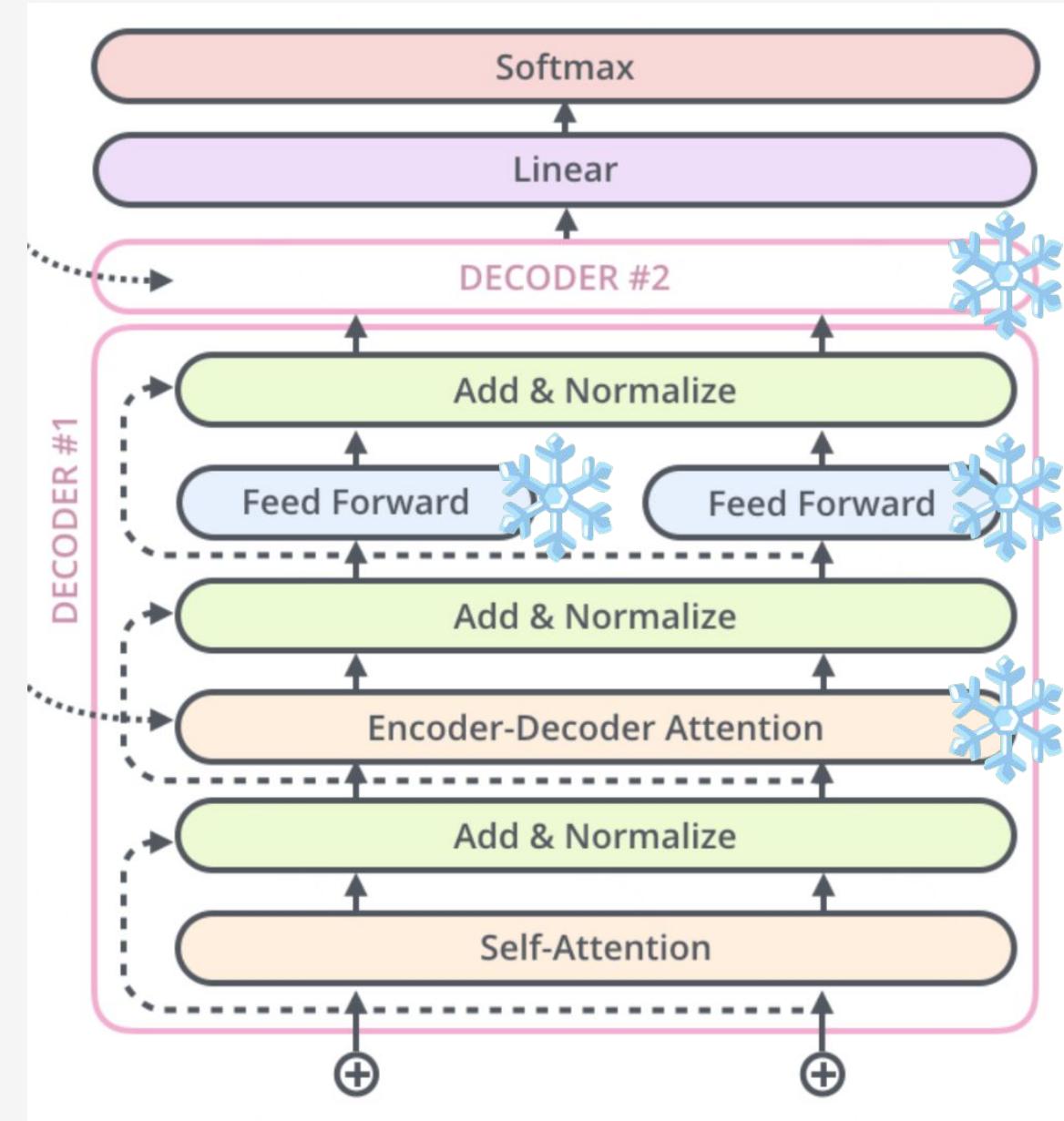
Частичное обучение или заморозка слоев позволяют изменять при дообучении только конкретные блоки (слои или даже параметры) модели.

Стандартные комбинации:

- ❖ Учится только голова модели;
- ❖ Обновление только верхних k слоёв;
- ❖ Верхним слоям — больший LR, нижним — меньший;
- ❖ Заморозка эмбеддингов, обучаем только трансформерные слои и голову.

**Плюсы:** экономия памяти/времени, уменьшение риска забывания.

**Минусы:** метрики могут быть ниже, чем при SFT.

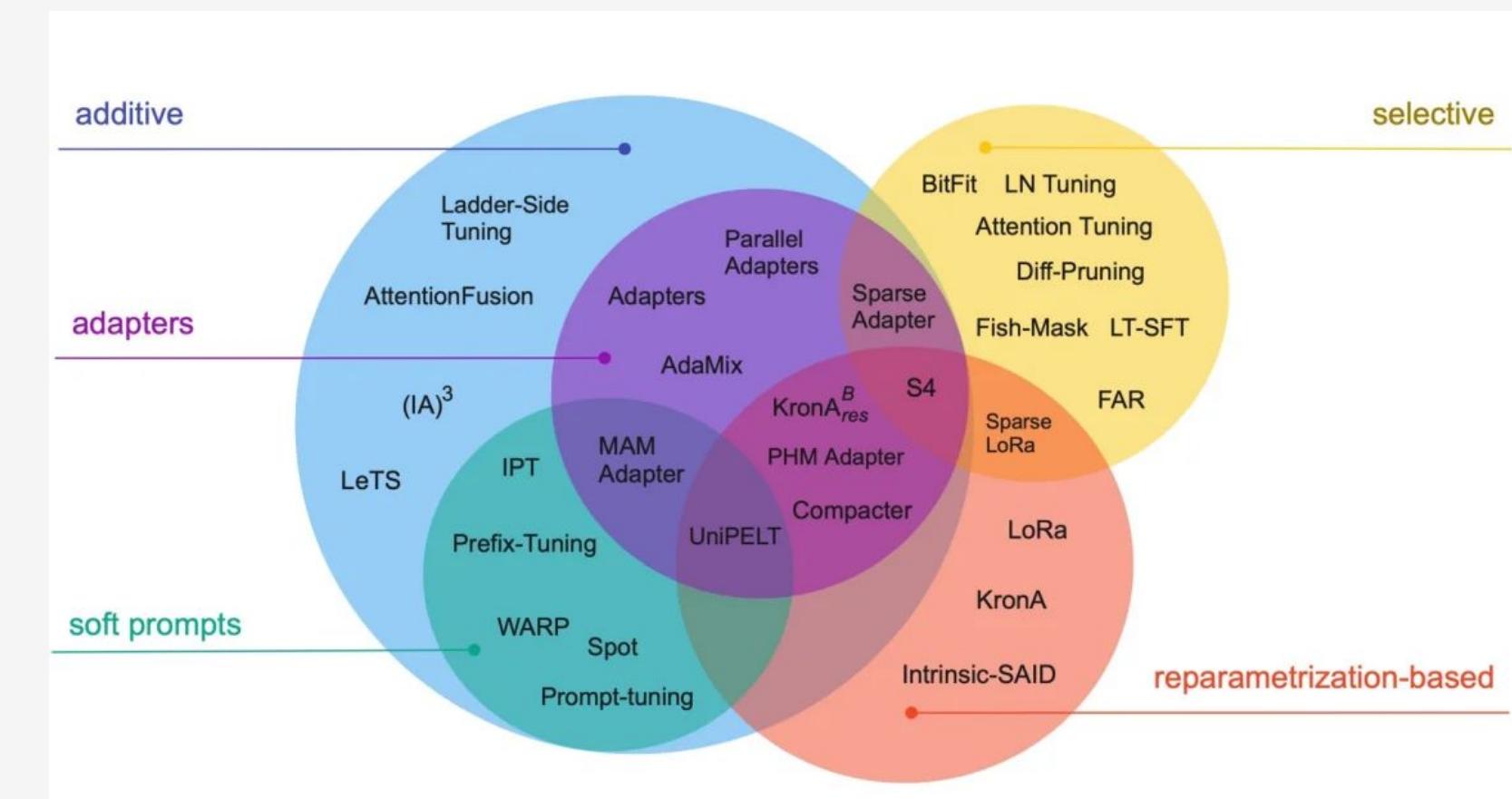


# PEFT

PEFT (Parameter-Efficient Fine-Tuning) - общее название для методов, которые изменяют лишь небольшую часть параметров (или добавляют небольшие модули). От заморозки отличается более сложным подходом к обучению.

Мы обсудим:

- Prompt Tuning
- LoRA (Low-Rank Adaptation)
- Adapters



# Prompt Tuning

Добавляем к модели обучаемый промпт:  
набор эмбеддингов, которые встраиваются в  
начало каждой последовательности.

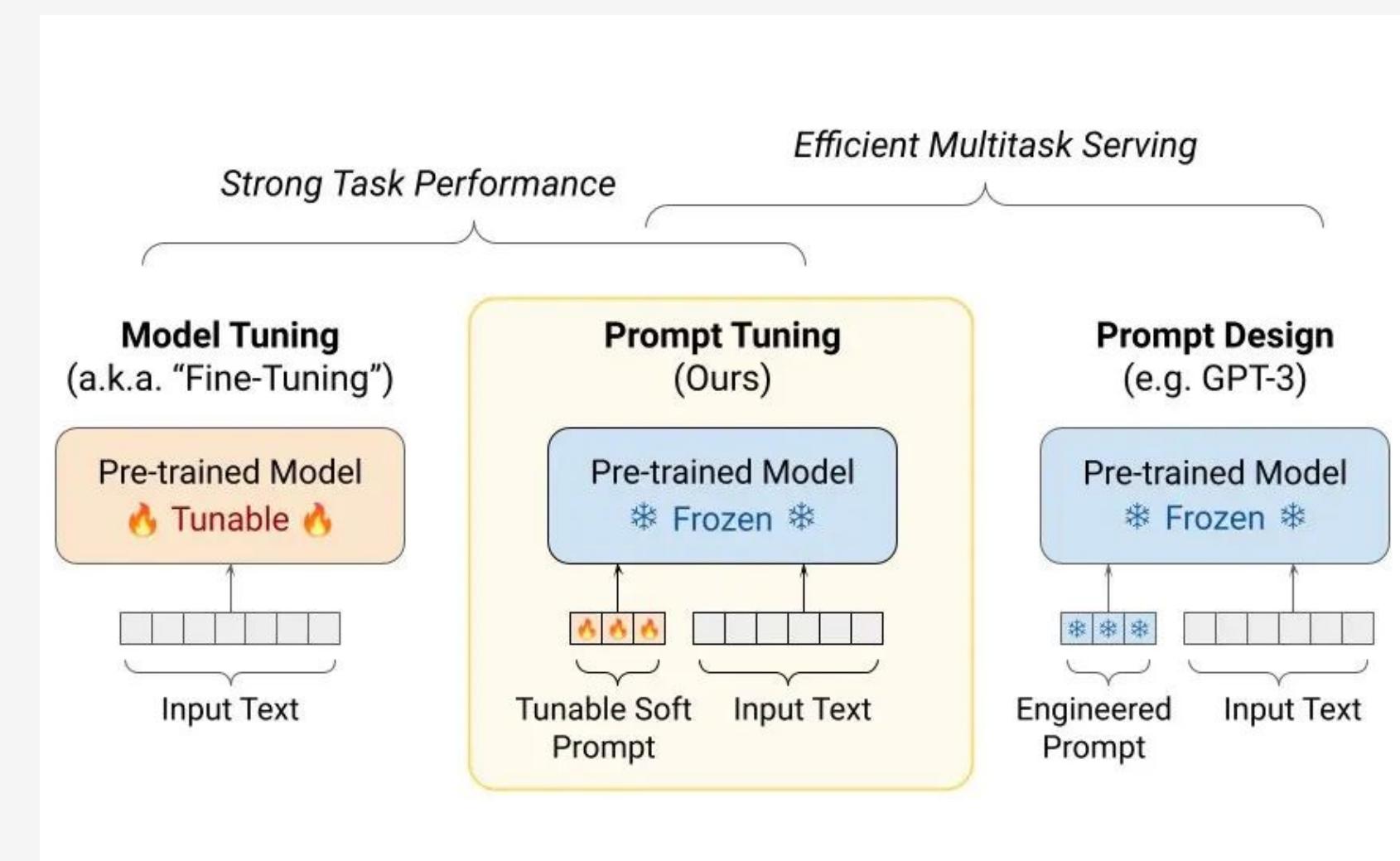
$[p_1, p_2, \dots, p_m, x_1, x_2, \dots, x_n]$   
обучаемый промпт

Плюсы:

- Можно иметь свой промпт под каждую задачу;
- Сильно гибче, чем написание промпта руками;
- Минимум ресурсов на обучение.

Минусы:

- Тратит токены контекста;
- Хуже качество, чем у адаптеров и LoRA.



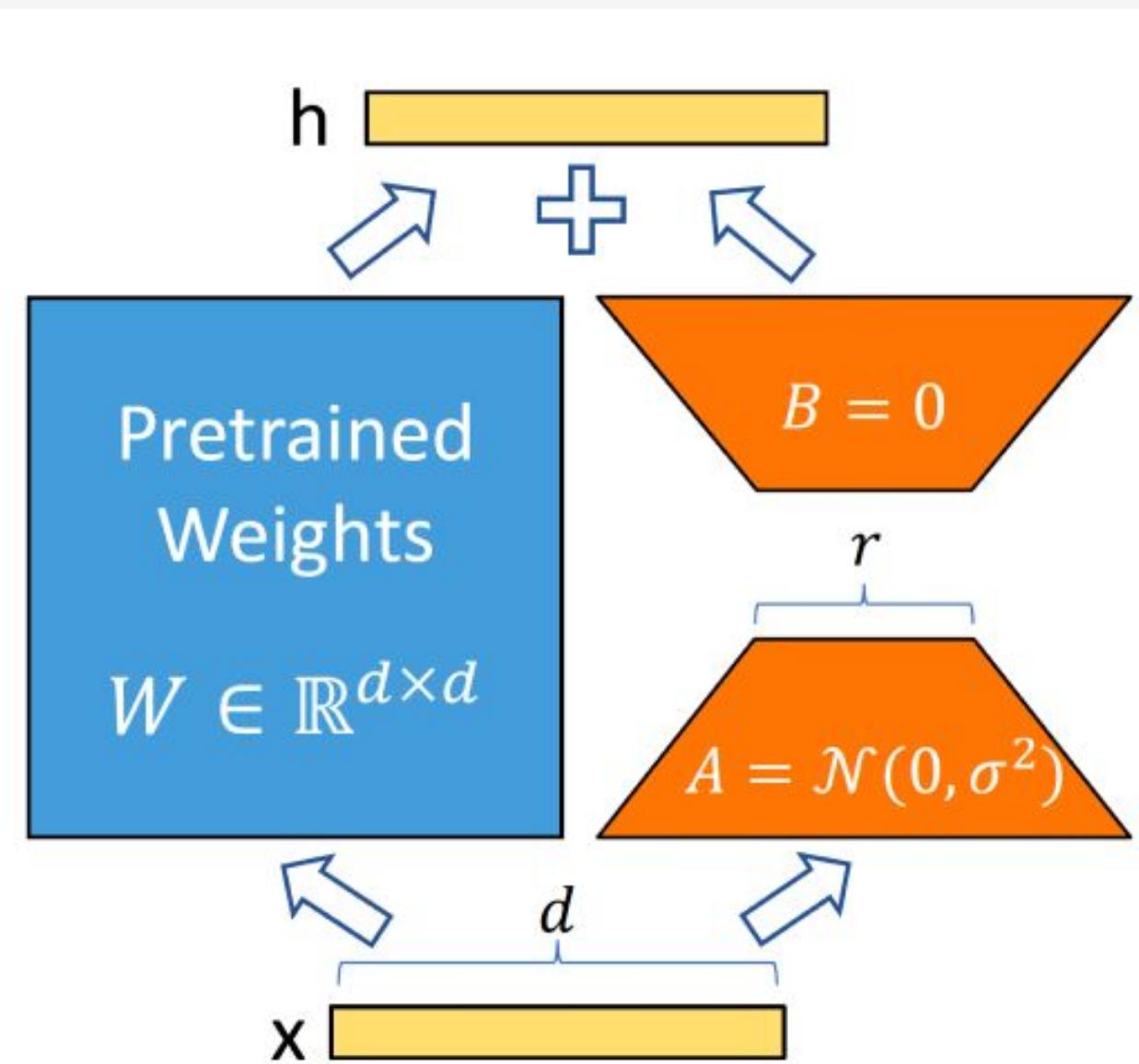
# LoRA

Вместо дообучения самой матрицы весов, мы обучаем для нее низкоранговое смещение (часто всего  $r$  от 4 до 64) - это дополнительные матрицы  $A$  и  $B$ , произведение которых потом прибавляется к реальным весам.

Такое смещение применяется к линейным проекциям:  $W_q$ ,  $W_k$ ,  $W_v$ ,  $W_o$ , реже к FF-слоям (там это сильно дороже).

Где выгода:

- В полноценном смещении для матрицы  $4096 \times 4096$  будет 16.7М параметров;
- В LoRA матрицах с рангом 8 будет суммарно  $4096 \times 8 \times 2 = 65536$  параметров. В 256 раз меньше!



# Adapters

Добавляем к модели маленькие FF-блоки - адаптеры, и тренируем только их, а не модель.

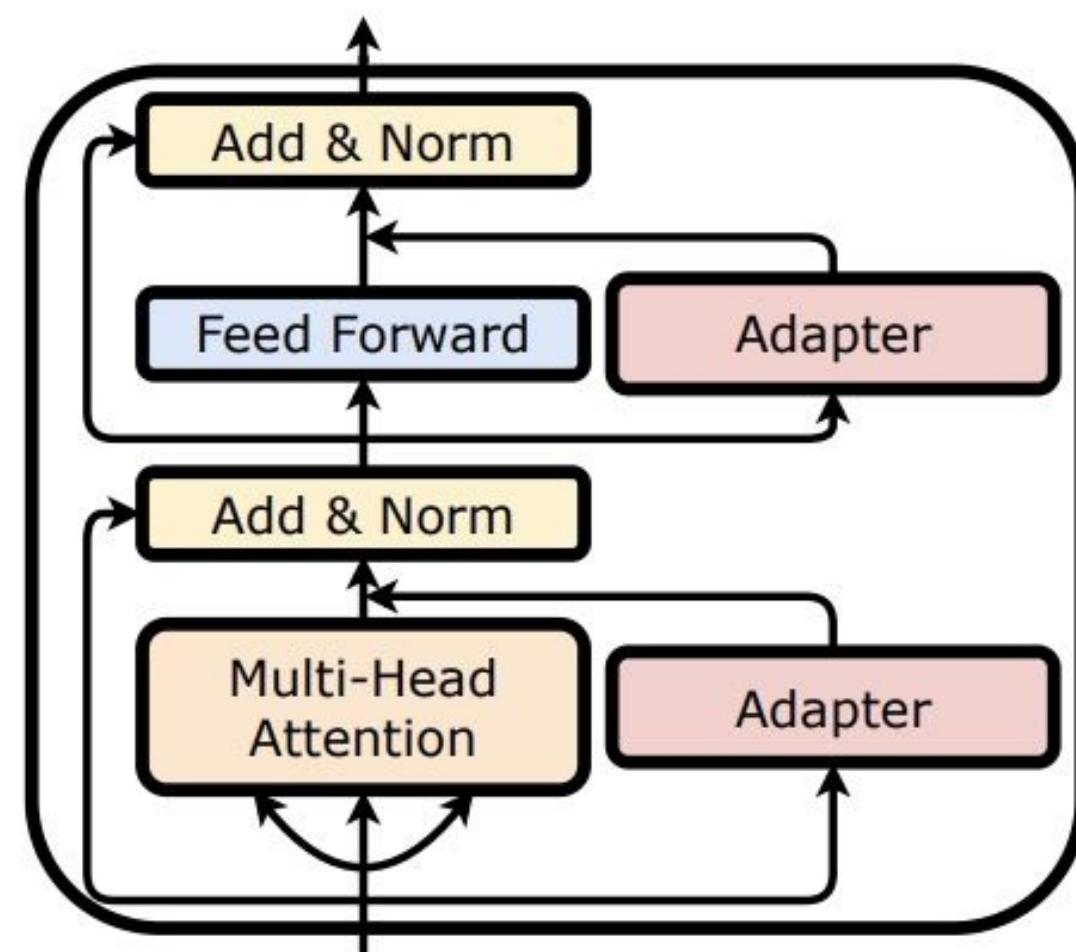
Плюсы:

- Сильно меньше весов для тренировки;
- Можно иметь разные адаптеры под разные задачи.

Минусы:

- Не мерджатся в модель без потерь (в отличие от LoRA);
- Как следствие, усложняют инференс и делают его дороже.

На практике LoRA-матрицы тоже почему-то часто называют адаптерами.



# Оптимизации

Кроме разных подходов к дообучению, еще есть разные аппаратные и программные трюки, которые делают LLM быстрее и дешевле:

- Gradient Accumulation: имитируем большой размер батча за счет усреднения градиентов с нескольких шагов;
- Mixed Precision: чаще всего используется BF16, где знак/мантийса/экспонента содержат 1/7/8 бит, что дает в 2 раза меньше памяти при меньшей точности, но без переполнения:

Формат	Мантийса	Экспонента
FP32	23	8
FP16	10	5 - переполняется
BF16	7	8

$$x = (-1)^s \cdot (1 + \text{мантийса}) \cdot 2^{\text{экспонента} - \text{bias}}$$

- Quantization (8-bit, 4-bit): храним веса в int8, int4.