**Оглавление**

[Общая информация 2](#_Toc201592636)

[LLM «под капотом» 3](#_Toc201592637)

[Шаги работы LLM 3](#_Toc201592638)

[Архитектура Transformer 4](#_Toc201592639)

[Программы для запуска моделей 5](#_Toc201592640)

[Дополнительные понятия 6](#_Toc201592641)

[Квантование 6](#_Toc201592642)

[GPU или CPU 7](#_Toc201592643)

[Inference 8](#_Toc201592644)

# Общая информация

**LLM** – Large Language Model – большая языковая модель.

**LLM** – это тип искусственного интеллекта, основанный на нейронных сетях, который обучен понимать и генерировать человеческий язык.

Это сложная структура из **параметров (весов)** и архитектуры, которая позволяет модели понимать и генерировать язык.

**Параметры – это числа**

* Это веса нейронной сети, которые определяют, как модель обрабатывает данные.
* Например, Llama 3 8B имеет 8 миллиардов параметров.
* Эти числа «запоминаются» моделью в процессе обучения и отвечают за то, какие закономерности модель узнаёт в тексте.

Когда мы скачиваем модель, мы, по сути, скачиваем набор параметров. А обучение этих моделей (накопление параметров/весов) было выполнено ранее на мощных компьютерах в течение довольно долгого времени.

Два основных типа моделей:

* **Open** – открытые

*Llama* (Large Language Model Meta AI) от Meta, *Gemma* от Google, *Mistral* от Mistral AI, *DeepSeek*, *Qwen*

* **Propriety** – закрытые / частные / проприетарные / запатентованные

*GPT* от Open AI, *Gemini* от Google, *Claude* от Anthropic

У открытых моделей Open LLM мы можем посмотреть параметры и веса (но далеко не всегда нам доступен обучающий код и обучающие алгоритмы модели).

У закрытых моделей мы не имеет доступа к параметрам, а можем использовать только предоставленный компанией владельцем АПИ для общения с моделью.

**LLM** – это программа, которая учится на огромном количестве текста из интернета, книг и статей.

Она запоминает, какие слова обычно идут друг за другом, и использует эти знания, чтобы предсказывать следующее слово в предложении.

# LLM «под капотом»

## Шаги работы LLM

Ты пишешь запрос, например:

"Расскажи о жизни животных в Африке."

1. Модель разбивает твой текст на части (слова или части слов) – это называется **токенизация**.
2. Модель анализирует контекст и смотрит, какие слова чаще всего появляются в таких ситуациях.
3. Генерирует ответ, выбирая по одному слову за раз, основываясь на том, что "учила" раньше.
4. Выдаёт результат – готовый текст, который кажется логичным и связным.

**Токены**

**Токен** – это минимальная часть текста, которую модель может обработать.

*Привет, как дела?* ---> *["Привет", ",", " как", " дела", "?"]*

**Токенизатор** – специальная программа, которая «переводит» текст в числа. Потому что модель не работает с буквами или целыми словами напрямую. Она преобразует токены в числа, чтобы с ними можно было работать *математически*.

Затем каждый токен превращается в число – **числовой вектор**

*["Привет", ",", " как", " дела", "?"]* ---> [1078, 12, 456, 982, 345]

Далее промт проходит через слои, где активно взаимодействует с весами (параметрами), хранящимися на этом слое.

Числовой вектор (представление токена) **умножается на веса на каждом слое**, и это – *основной принцип работы нейронных сетей*, включая LLM.

**Слои**

Каждый **слой в LLM** – это отдельный набор параметров (весов), через которые проходит входной вектор.

|  |  |
| --- | --- |
| **Первые слои** | Узнают базовые закономерности: части слова, грамматика |
| **Средние слои** | Понимают контекст: кто, что делает, о чём речь |
| **Последние слои** | Формируют смысл: делают вывод, пишут ответ |

На каждом слое вектор умножается на соответствующие веса.

Проходит через *функцию активации* (например, ReLU или GELU), чтобы добавить «нелинейность».

И передается на следующий слой, где с ним происходит тоже самое

## Архитектура Transformer

В современных LLM (вроде Llama, Mistral, Falcon и др.) используется Transformer – специальная архитектура, где:

* Есть **слои внимания (attention)** – они сравнивают все слова друг с другом, чтобы понять контекст.
* Есть **проекционные слои** – умножают векторы на веса для обработки информации.
* Все эти операции – умножение векторов на матрицы весов.

# Программы для запуска моделей

**Llama.cpp** – разработанная компанией Meta программа на C++ для запуска языковых моделей. Низкоуровневая работа с LLM

**LM Studio** – коммерческая обертка над Llama.cpp с приятным UI

**Ollama** – open-source программа, обертка над Llama.cpp без UI, но с удобной CLI

# Дополнительные понятия

## Квантование

**Квантование Quantization** — это метод оптимизации нейросетей, в том числе языковых моделей (LLM), который позволяет уменьшить размер модели и ускорить её работу, при этом сохраняя большую часть качества.

Это делается для того, чтобы ускорить модель и сделать возможным ее запуск даже на слабых устройствах без GPU

Все модели хранят свои веса (параметры) в виде чисел.

Изначально эти числа часто имеют тип **float32** или **float16** — то есть каждое число занимает **4 байта памяти** или **2 байта памяти** соответственно.

При квантовании мы понижаем точность этих чисел: до *float16 (2 байта)*, или даже до *int8 (1 байт = 8 бит)*, *int4 (0.5 байта = 4 бит)*

Например, модель с 7 миллиардами параметров:

В *float32* занимает около *28 ГБ*, в *int4* занимает около *3.5 ГБ.*

**Виды квантования (по точности)**

|  |  |  |
| --- | --- | --- |
| **Тип** | **Описание** | **Потеря качества** |
| FP32 → FP16 / BF16 | Понижение точности с 32 до 16 бит | Очень маленькая |
| INT8 | Целочисленное представление (8 бит) | Небольшая |
| INT4 | Еще более агрессивное квантование (4 бита) | Средняя, но допустимая |
| GPTQ, AWQ, GGUF и др | Специальные методы квантования под конкретные задачи | Разная, зависит от метода |

## GPU или CPU

**GPU – Graphics Processing Unit** – Графический процессор

Изначально создавался для обработки графики (например, игр), но сегодня широко используется в машинном обучении и ИИ, потому что отлично справляется с параллельными вычислениями.

Обучение и запуск (инференс) LLM требует огромного количества математических операций.

GPU может выполнять тысячи операций **одновременно** (параллельно), а CPU делает это последовательно.

|  |  |  |
| --- | --- | --- |
| **Характеристика** | **CPU** | **GPU** |
| Подходит для | Тестирование, мелкие модели | Все типы моделей |
| Скорость генерации | Медленная (долгие ответы) | Быстрая (доли секунды) |
| Поддержка квантования | Ограниченная | Полная |
| Цена | Дешевле | Дороже |
| Энергопотребление | Выше | Ниже |

## Inference

**Генерация** – **Inference** – процесс взаимодейтсвия с моделью: ввод промта, обработка, вывод ответа

**temperature**, **top\_k**, **top\_p** и **min\_p** – это ключевые параметры генерации текста в LLM, которые позволяют управлять разнообразием, стабильностью и качеством выходных данных.

LM генерирует токен-кандидаты



И эти параметры определяют то, какие именно кандидаты будут выбраны при генерации ответа

**Temperature** – управляет случайностью: чем выше, тем более творческий/непредсказуемый вывод. Иными словами, вариативность ответов, если мы будем повторять промт

*temperature = 0* – детерминированный вывод (постоянный результат)

*temperature = 1* – более случайные, разнообразные ответы

В примере выше:

При температуре близкой к 0, у токена blue в примере ниже вероятность поднимется до 90-95% (то есть постоянно будет один и тот же результат), а при приближении температуры к 1 вероятность всех кандидатов будет уравниваться и, например, стремиться к 25% (если кандидатов четыре)

**top\_k**

Ограничивает выбор следующего токена только k наиболее вероятными вариантами

Если в примере выше top\_k будет равно 1, то всегда будет выбираться blue, т.к. у нас может быть только один (top\_k=1) кандидат и берется самый вероятный. Если top\_k будет равен 2, то выбор будет происходить между blue и visible, как двумя наиболее вероятными кандидатами и так далее

**top\_p**

Это параметр, который говорит модели:

Выбирай следующий токен только из тех вариантов, суммарная вероятность которых составляет не менее **p** от общей вероятности (начиная перебор от самого вероятного к менее вероятному)

В примере выше для *top\_p = 0.5* (то есть суммарная вероятность 50%) выбор будет происходить между blue (45%) и visible (21%)

Для *top\_p = 0.7* (то есть суммарная вероятность 70%) выбор будет происходить между blue (45%), visible (21%) и clear (12%)

**min\_p**

Задает *минимальную вероятность*, которую должен иметь следующий токен (слово или часть слова), чтобы его можно было выбрать для добавления в сгенерированный текст.

В примере выше для min\_p = 0.2 выбор будет происходить только из blue (45%) и visible (21%), остальные токены будут отброшены

Зачем нужен **min\_p**? Какие проблемы он решает?

1. Предотвращение "галлюцинаций" и бессвязного текста. Отсечь маловероятные варианты и повысить связность и согласованность генерируемого текста
2. Контроль над креативностью

Например, при высоком *min\_p*. текст будет очень предсказуем, формальным и часто скучным, но он хорошо подходит для документов

1. Управление случайностью.

Низкий *min\_p* позволяет получать более неожиданные ответы