

ИНТЕЛЛЕКТ ➔

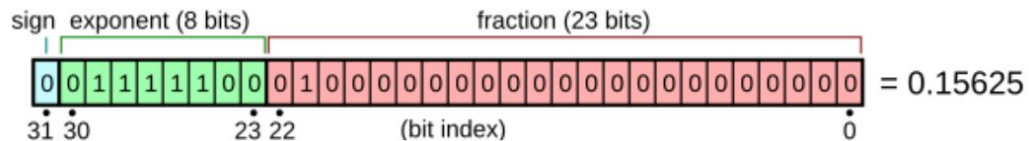
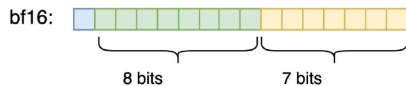
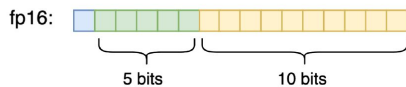
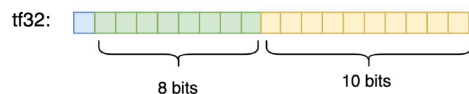
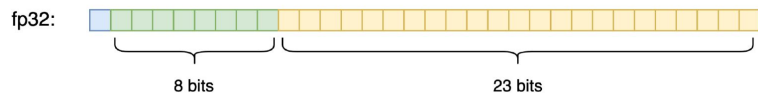
# Inference LLM

к.ф.-м.н. Тихомиров Михаил Михайлович

научный сотрудник  
НИВЦ МГУ имени М. В. Ломоносова

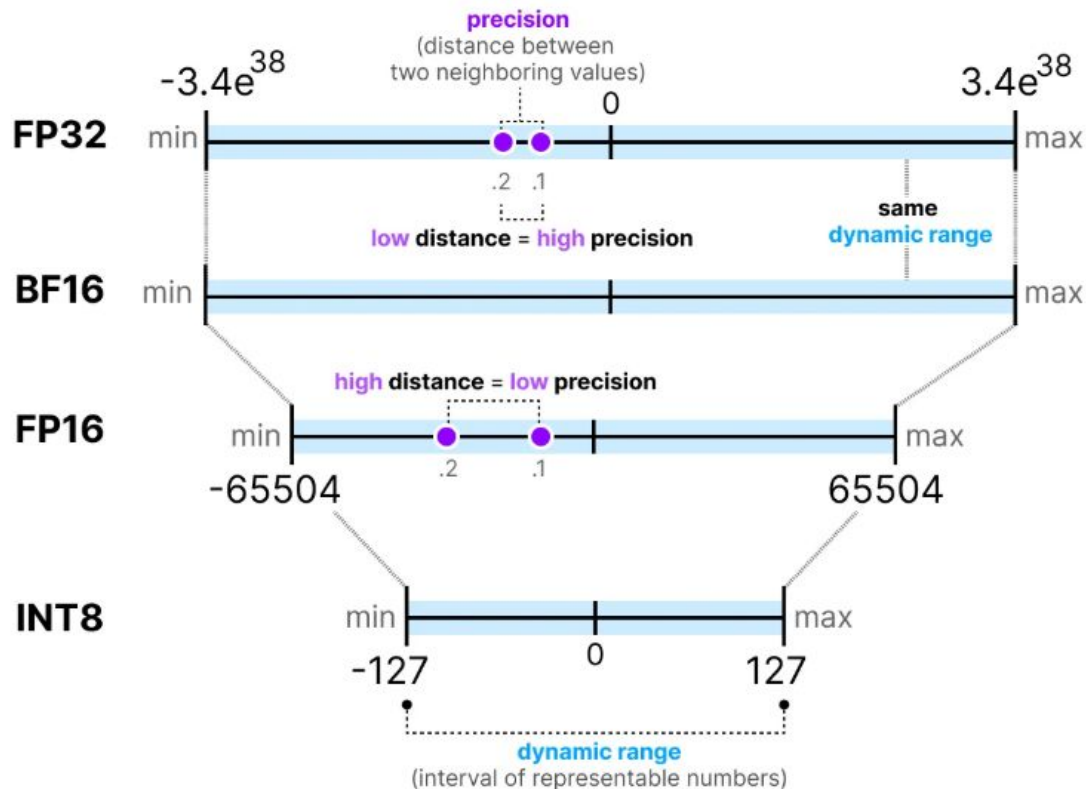
# Квантизация

# Представление чисел



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

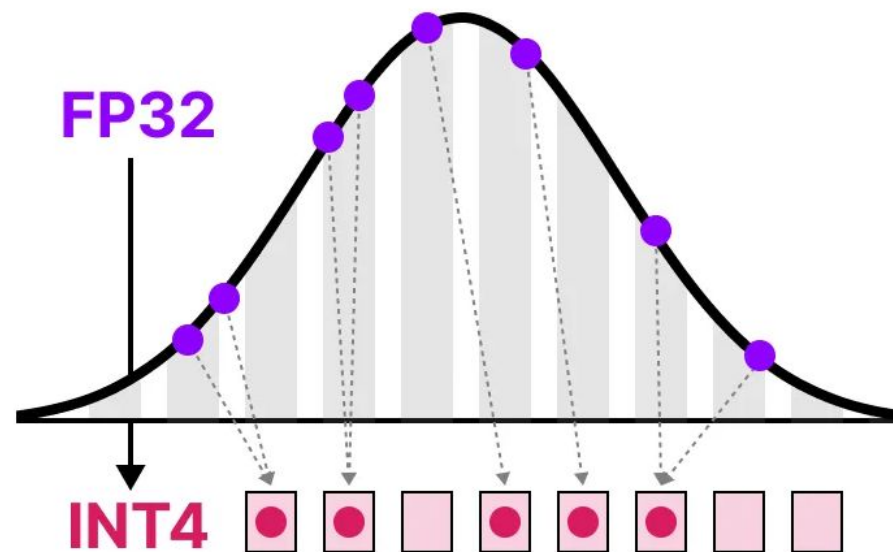
# Квантизация: диапазон значений



# Квантизация



	Scalar	Vector	Matrix	Tensor
Quantized	1	$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\begin{bmatrix} [1 & 2] & [3 & 2] \\ [1 & 7] & [5 & 4] \end{bmatrix}$
High Precision	1.3	$\begin{bmatrix} 1.4 \\ 2.7 \end{bmatrix}$	$\begin{bmatrix} 1.8 & 2.2 \\ 3.7 & 4.2 \end{bmatrix}$	$\begin{bmatrix} [1.1 & 2.7] & [3.75 & 2.7] \\ [1.7 & 7.9] & [5.31 & 4.9] \end{bmatrix}$



# Absmax



Формула:

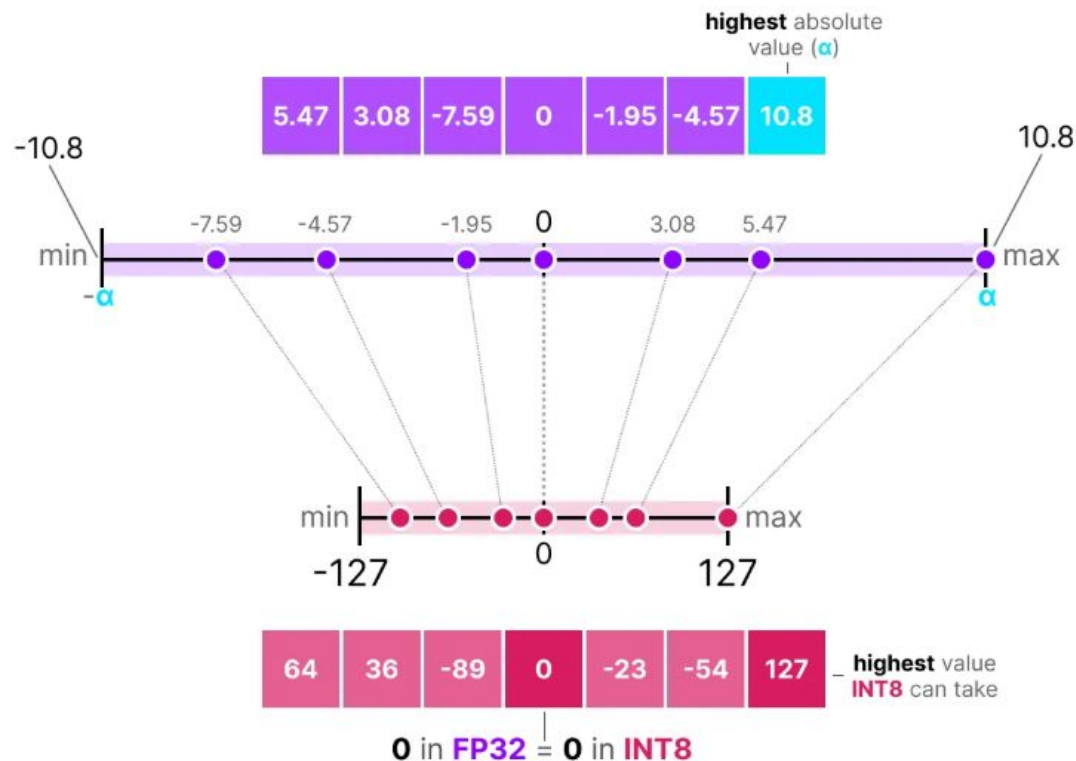
$$s = \frac{127}{\max_{i,j} |X_{i,j}|}$$

Квантизация:

$$X_q = \text{round}(s \cdot X)$$

Деквантизация:

$$X_{dq} = \frac{X_q}{s}$$



# Zero-point



## Формула:

$$\begin{cases} s \min_{i,j} X_{i,j} + z = -128, \\ s \max_{i,j} X_{i,j} + z = 127 \end{cases} \iff \begin{cases} s = \frac{255}{\max_{i,j} X_{i,j} - \min_{i,j} X_{i,j}}, \\ z = -s \min_{i,j} X_{i,j} - 128 \end{cases}$$

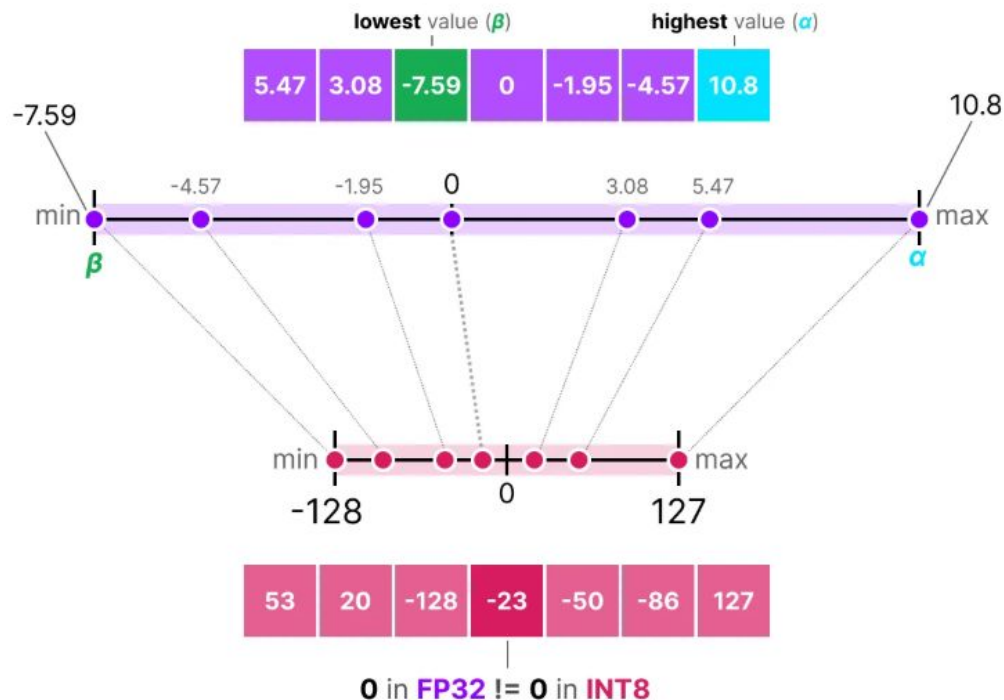
$$\tilde{z} = \text{round}(z)$$

## Квантизация:

$$X_q = \text{round}(s \cdot X + \tilde{z}),$$

## Деквантизация:

$$X_{dq} = \frac{X_q - \tilde{z}}{s}$$



# Absmax vs zero-point: пример



$$X = \begin{pmatrix} 1.5 & 0 & -1.1 \\ 2 & 0.5 & -0.1 \\ 1.2 & 1.7 & -0.5 \end{pmatrix}$$



**absmax**

$$X_q = \begin{pmatrix} 95 & 0 & -70 \\ 127 & -2 & 36 \\ 76 & 108 & -32 \end{pmatrix}$$



$$X_{dq} = \begin{pmatrix} 1.496 & 0 & -1.102 \\ 2 & 0.504 & -0.094 \\ 1.197 & 1.701 & -0.504 \end{pmatrix}$$

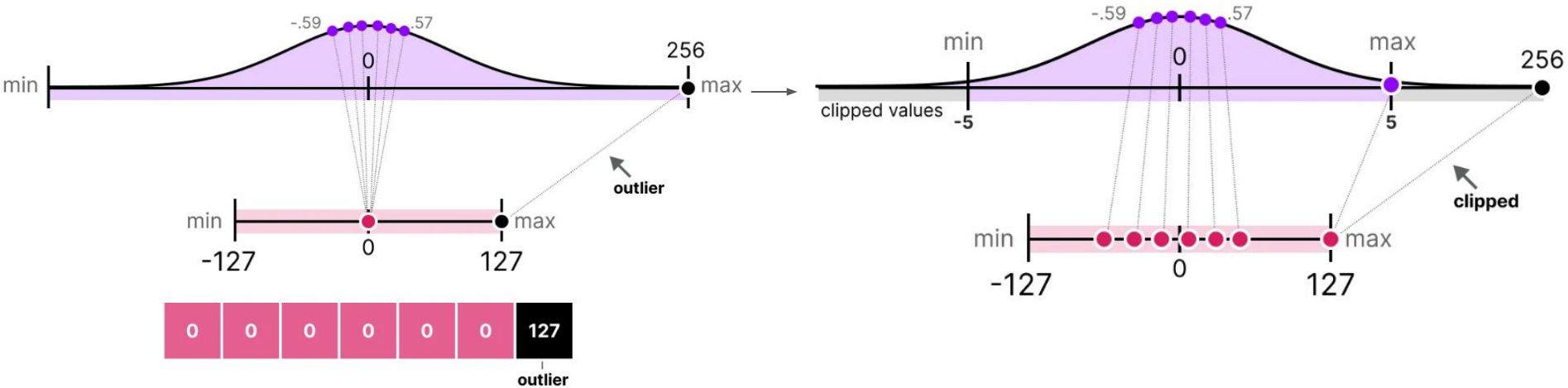
**zero-point**

$$X_q = \begin{pmatrix} 85 & -38 & -128 \\ 127 & -3 & -46 \\ 61 & 102 & -79 \end{pmatrix}$$

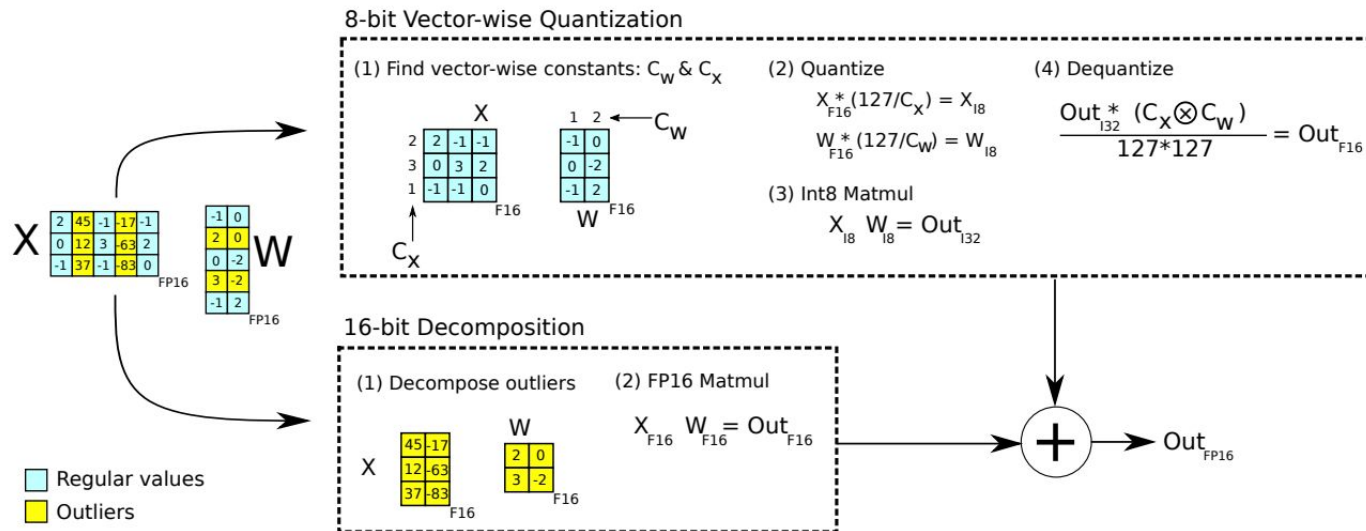


$$X_{dq} = \begin{pmatrix} 1.495 & 0 & -1.094 \\ 2.006 & 0.498 & -0.097 \\ 1.204 & 1.702 & -0.498 \end{pmatrix}$$

# Выбросы



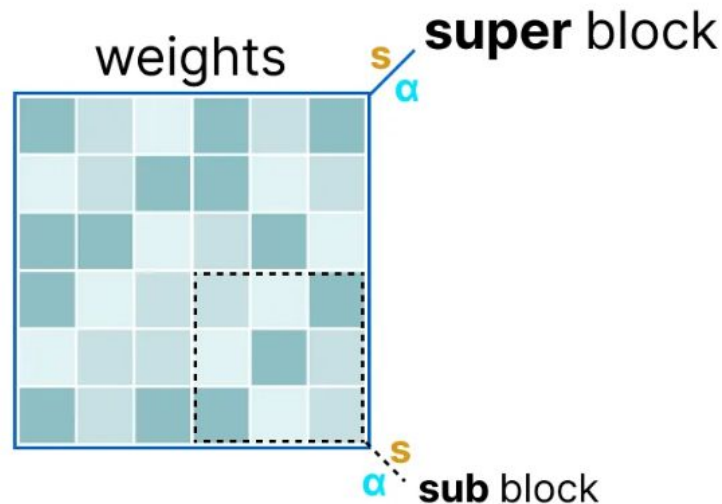
- vector-wise  
квантизация
- отдельная  
обработка  
выбросов





- Веса нейросети распределены около 0 и нормально  $\rightarrow$  можно нормализовать к среднему = 0 и диапазону  $[-1;1]$
- Рассмотрим 4-бит квантизацию, выделив на диапазоне  $[-1;1]$  16 значений:
  - $[-1, -0.8667, -0.7333, \dots, -0.0667, 0.0667, \dots, 0.7333, 0.8667, 1]$
- Находим для нашего числа ближайшее значение в векторе, его индекс и будет квантизированным значением

- Слои разбиваются на суперблоки
- Каждый блок разбивается на вложенные блоки (sub block)
- Каждый вложенный блок квантизуется abtmax методом
- NO scale factor каждого вложенного блока также квантизуется на уровне суперблока



$$X_{\text{quantized}} = S \cdot X$$

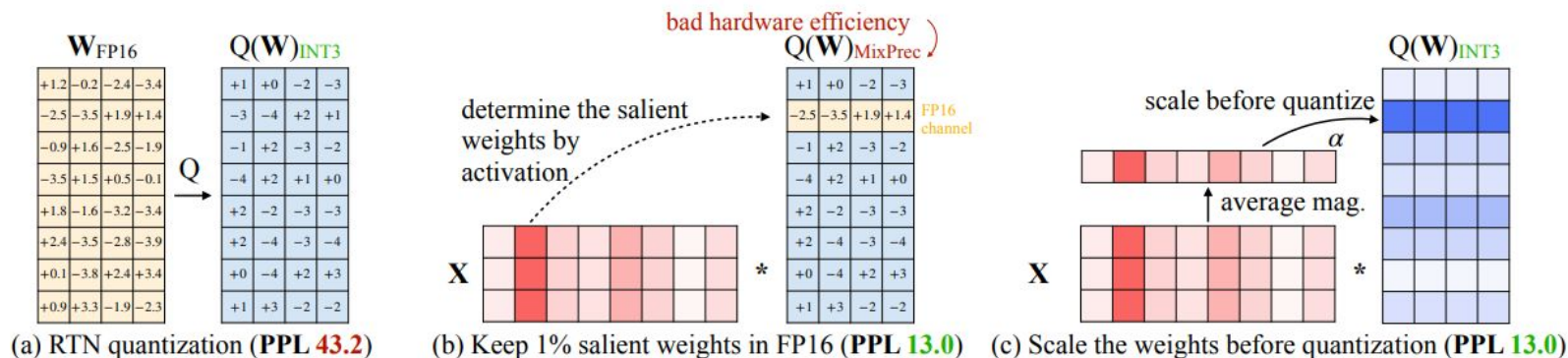
$$X_{\text{quantized}} = \underbrace{S_{\text{sub}}}_{\substack{\text{quantized using} \\ S_{\text{super}}}} \cdot X$$

Используется в llama.cpp и его  
клонах!

# AWQ: Activation-aware Weight Quantization



- Собирает статистику активаций на некотором небольшом корпусе
- “Все животные равны, но некоторые равнее (с)” - не все веса одинаково важны, причем разница очень существенная.



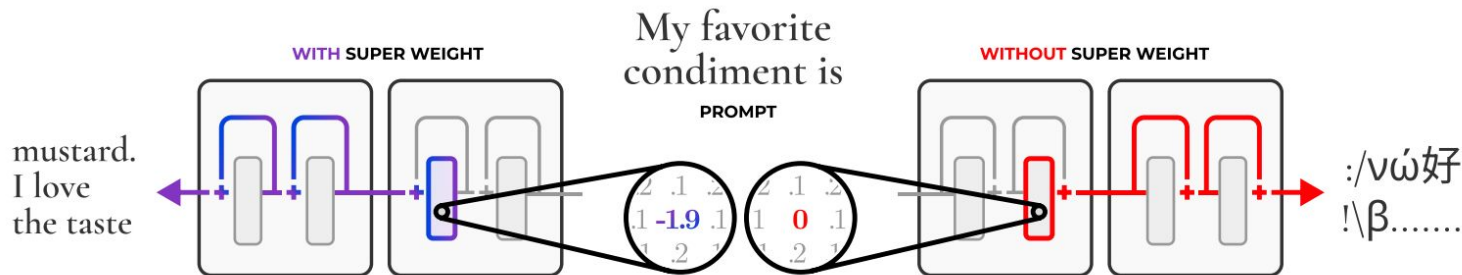
Используется в vLLM.

# Феномен “супер весов”

Супер веса - веса, которые существенно влияют на поведение модели (одно число в конкретной матрице!)

В исследованных моделях подобных весов около десятка!

Обычно в первых слоях в MLP.

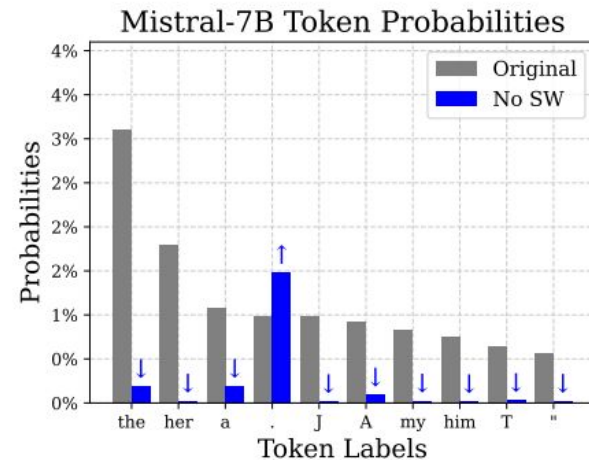
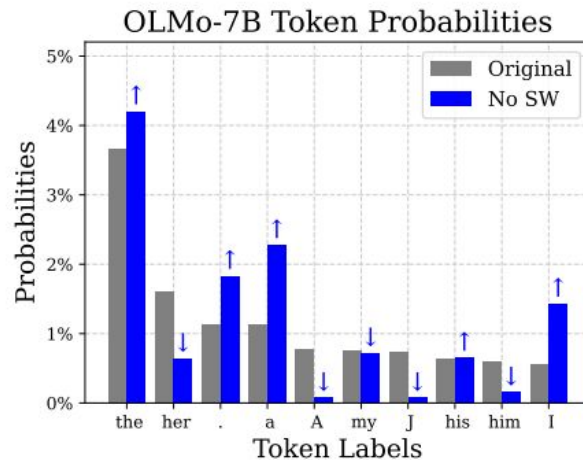
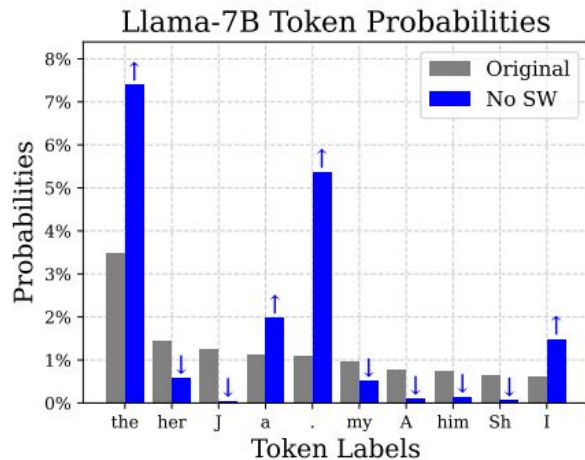


# Феномен “супер весов”: деградация качества



<b>Llama-7B</b>	<b>Arc-c</b>	<b>Arc-e</b>	<b>Hella.</b>	<b>Lamb.</b>	<b>PIQA</b>	<b>SciQ</b>	<b>Wino.</b>	<b>AVG</b>	<b>C4</b>	<b>Wiki-2</b>
Original	41.81	75.29	56.93	73.51	78.67	94.60	70.01	70.11	7.08	5.67
Prune SW	19.80	39.60	30.68	0.52	59.90	39.40	56.12	35.14	763.65	1211.11
Prune Non-SW	41.47	74.83	56.35	69.88	78.51	94.40	69.14	69.22	7.57	6.08
Prune SW, +SA	26.60	54.63	56.93	12.79	67.95	61.70	70.01	50.09	476.23	720.57

# Феномен “супер весов”: влияние на стоп слова



# Инференс

# Can i run it LLM?



## Can you run it? LLM version

### Information

- GPU information comes from [TechPowerUp GPU Specs](#)
- Mainly based on [Model Memory Calculator by hf-accelerate](#) using `transformers` library
- Inference is calculated following [EleutherAI Transformer Math 101](#), where is estimated as

$$\text{Memory}_{\text{Inference}} \approx \text{Model Size} \times 1.2$$

- For LoRa Fine-tuning, I'm assuming a **16-bit** dtype of trainable parameters. The formula (in terms of GB) is

$$\text{Memory}_{\text{LoRa}} \approx \left( \text{Model Size} + \# \text{ trainable Params}_{\text{Billions}} \times \frac{16}{8} \times 4 \right) \times 1.2$$

### [NousResearch/Meta-Llama-3-8B-Instruct](#) (7.6B)

int4 int8 float16/bfloat16 float32

int4 refers to models in GPTQ-4bit, AWQ-4bit or Q4\_0 GGUF/GGML

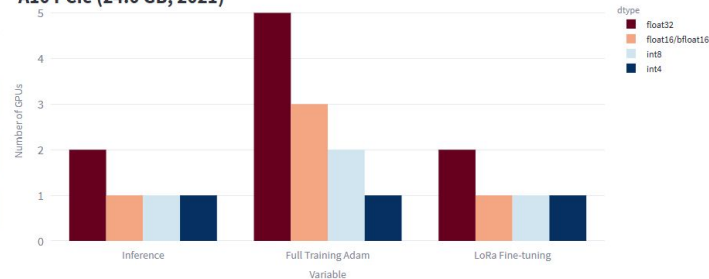
✓ You require 1 GPUs for Inference

✓ You require 1 GPUs for Full Training Adam

✓ You require 1 GPUs for LoRa Fine-tuning (2.0%)

	float32	float16/bfloat16	int8	int4
Total Size (GB)	27.96	13.98	6.99	3.49
Inference (GB)	33.55	16.77	8.39	4.19
Training using Adam (GB)	111.83	55.92	27.96	13.98
LoRA Fine-Tuning (GB)	34.99	18.22	9.83	5.63

### Number of GPUs required for A10 PCIe (24.0 GB, 2021)



# Через что обычно “генерируют” с LLM



- Huggingface + transformers + pytorch
  - “Нативное” решение, низкая эффективность
- VLLM
  - Один из лучших фреймворков для инференса с нацеленностью на высокую пропускную способность
- SGLang
  - Аналог VLLM, где-то лучше, где-то хуже
- llama.cpp (или ее клоны)
  - llama



# Какие характеристики оценивают

- Занимаемая память
- Токен/секунда
- Пропускная способность при “нагрузке”
  - Обработка батчами (Batch inference)
  - Токен/секунда или RPS (запрос в секунду)
- Время до первого токена
- Поддерживаемые квантизации
- Возможность multigpu/multinode
- Возможность сru offload

# HF + transformers: инициализация



```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer, GenerationConfig

MODEL_NAME = "RefalMachine/RuadaptQwen2.5-1.5B-instruct"
model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    load_in_8bit=False,
    torch_dtype=torch.bfloat16,
    device_map="cuda:0",
    attn_implementation='flash_attention_2'
)
model.eval()

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
generation_config = GenerationConfig.from_pretrained(MODEL_NAME)
generation_config.max_new_tokens = 512
generation_config.temperature = 0.3
generation_config.repetition_penalty = 1.05
```

# HF + transformers: генерация



```
import time
inputs = ["Напиши функцию для генерации ответа через LLM, используя transformers."]
for query in inputs:
    prompt = tokenizer.apply_chat_template([
        {"role": "user",
         "content": query}
    ], tokenize=False, add_generation_prompt=True)

    data = tokenizer(prompt, return_tensors="pt", add_special_tokens=False)
    data = {k: v.to(model.device) for k, v in data.items()}

    s = time.time()
    output_ids = model.generate(**data, generation_config=generation_config)[0]
    output_ids = output_ids[len(data["input_ids"][0]):]
    output = tokenizer.decode(output_ids, skip_special_tokens=True).strip()
    gen_time = time.time() - s
    print(len(output) / gen_time)
    print(output)
```

# HF + transformers: пример результата



167.16134602604677

Для генерации ответа через LLM (Language Model) с использованием библиотеки `transformers` из PyTorch, вам нужно выполнить следующие шаги:

1. **Установите необходимые библиотеки**:

```
```bash
pip install transformers torch
```
```

2. **Создайте функцию генерации ответа**:

```
```python
from transformers import AutoModelForCausalLM, AutoTokenizer

def generate_response(prompt):
    # Устанавливаем модель и токенайзер
    model_name = "EleutherAI/gpt-neo-125M"
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForCausalLM.from_pretrained(model_name)

    # Предлагаемый запрос
    input_text = prompt

    # Преобразуем текст в токены
    inputs = tokenizer.encode(input_text, return_tensors='pt')

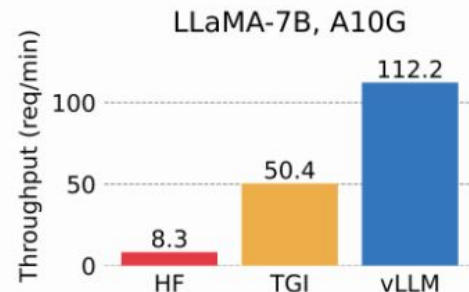
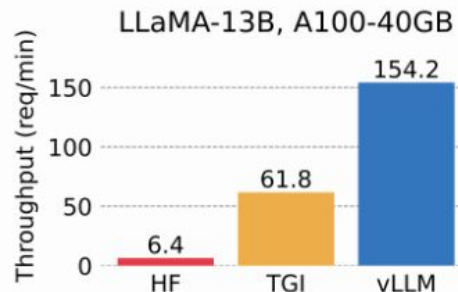
    # Генерируем ответ
    with torch.no_grad():
        outputs = model.generate(inputs, max_length=50, num_return_sequences=1)

    # Переводим ответ обратно в текст
    response_text = tokenizer.decode(outputs[0], skip_special_tokens=True)

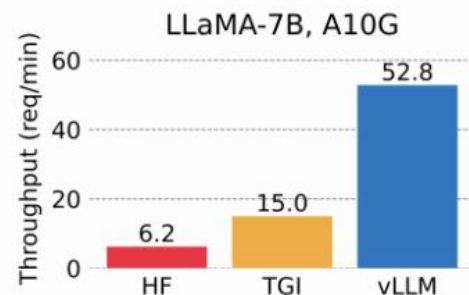
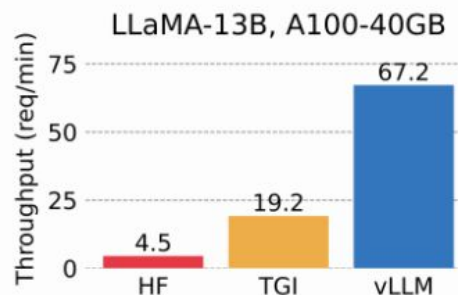
    return response_text

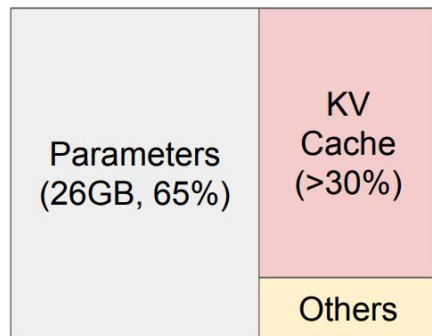
# Пример использования
prompt = "Какой будет следующий шаг в разработке проекта?"
print(generate_response(prompt))
```
```

- Одно из главных узких мест инференса LLM - память
- В частности классический attention + kv cache потребляет много (1.7GB for a single sequence in LLaMA-13B)
- Реальное ускорение не такое большое, как на графике

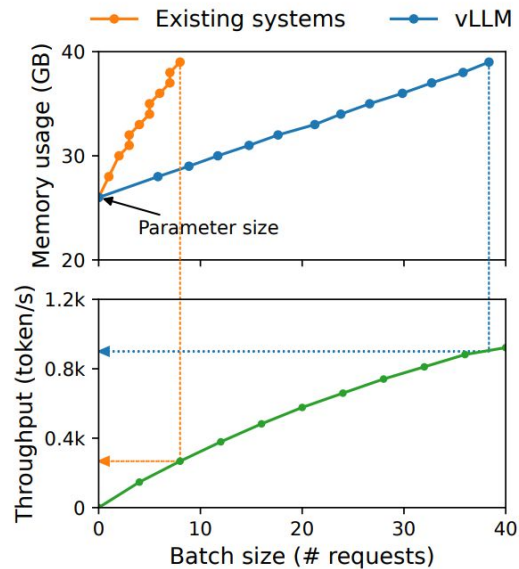


Serving throughput when each request asks for *one output completion*. vLLM achieves 14x - 24x higher throughput than HF and 2.2x - 2.5x higher throughput than TGI.





NVIDIA A100 40GB



# KV cache

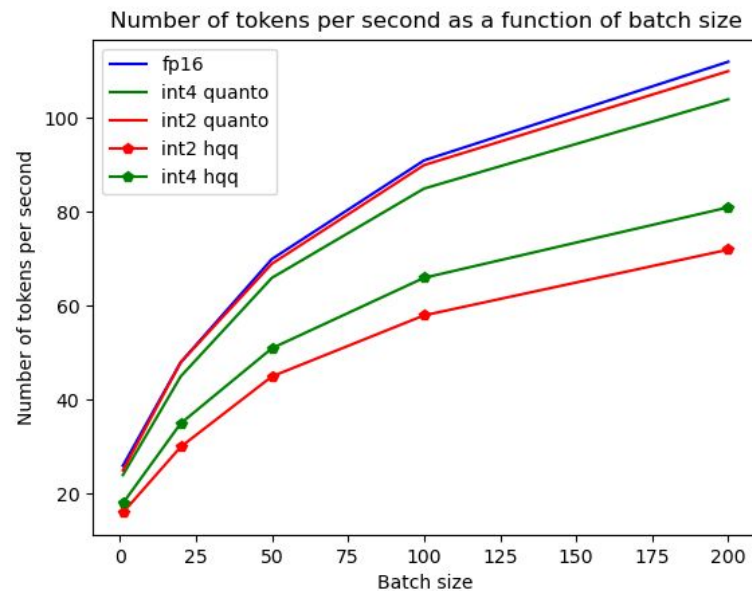
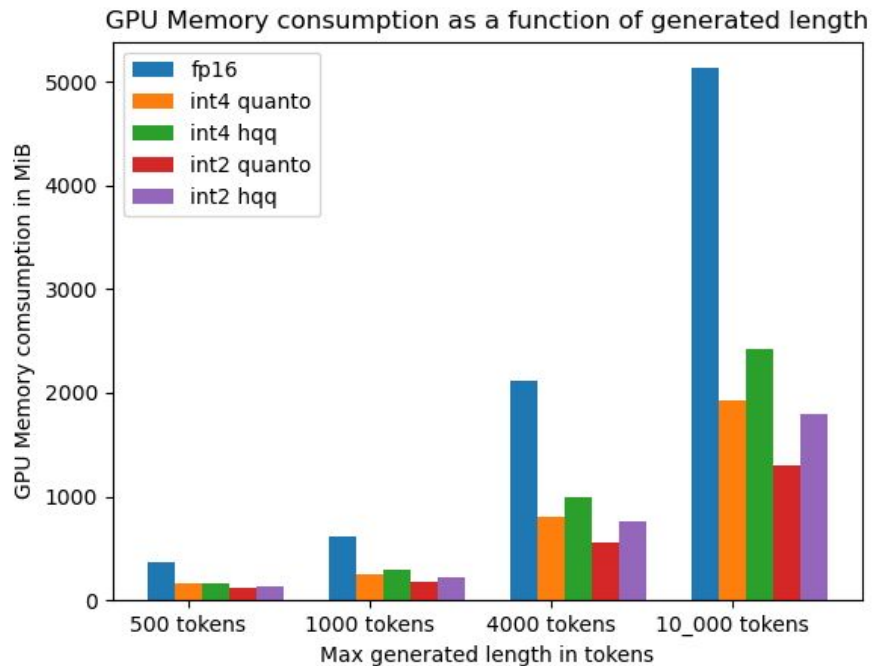


- Мы генерируем токен за токеном, высчитывая на каждом слое  $QKV \dots$
- $K$  и  $V$  не меняются для будущих токенов, так как зависят только от прошлых, и они нужны для расчета будущих токенов
- Их можно закешировать!

Для одного токена требуется байт (в fp16)

$$2 \cdot 2 \cdot n_{\text{layers}} \cdot n_{\text{heads}} \cdot d_{\text{head}}$$

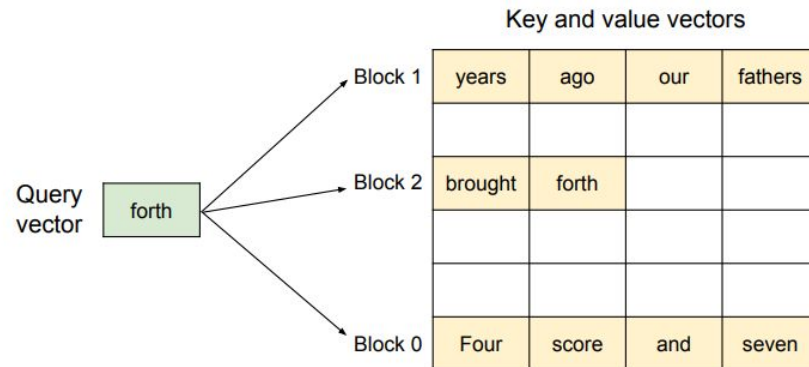
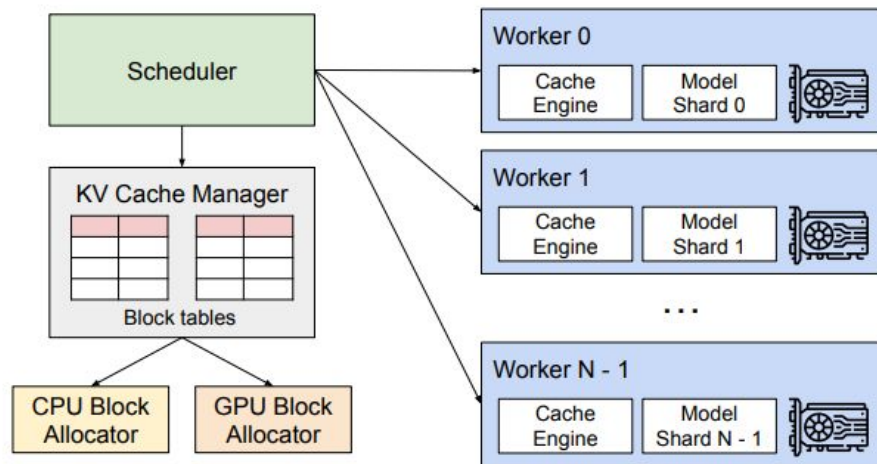
# KV cache: квантизированный





- Одна из основных фиш vLLM
- Разбивает kv-cache на части, эффективно их хранит и обращается по аналогии с страничной организацией памяти в ОС
- При одинаковых префиксах (промптах) и множественном декодировании, для разных последовательностей общая часть хранится только в едином экземпляре
- Во время расчета внимания, PagedAttention определяет и извлекает разные блоки KV-кеша по отдельности.

# Paged Attention



# VLLM: инициализация



```
from vllm import LLM, SamplingParams
from transformers import AutoTokenizer

MODEL_NAME = "RefalMachine/RuadaptQwen2.5-1.5B-instruct"
sampling_params = SamplingParams(
    temperature=0.01,
    top_p=0.9,
    top_k=40,
    max_tokens=1000,
    repetition_penalty=1.0,
)
llm = LLM(
    model=MODEL_NAME,
    max_seq_len_to_capture=4096,
    max_model_len=4096
)
tokenizer = llm.get_tokenizer()
```



```
import time
inputs = ["Напиши функцию для генерации ответа через LLM, используя transformers."]
for query in inputs:
    messages = [{
        "role": "user",
        "content": query
    }]
    prompt = tokenizer.apply_chat_template(
        messages, tokenize=True, add_generation_prompt=True
    )
    s = time.time()
    outputs = llm.generate(prompt_token_ids=[prompt], sampling_params=sampling_params)
    gen_time = time.time() - s
    for output in outputs:
        generated_text = output.outputs[0].text
        print(len(generated_text) / gen_time)
        print(generated_text)
```

# VLLM: пример результата



866.654579678111

Для генерации ответа через LLM (Language Model) с использованием библиотеки `transformers` из PyTorch, вам нужно выполнить следующие шаги:

1. **\*\*Установите библиотеки\*\***:

```
```bash
pip install transformers torch
```
```

2. **\*\*Создайте функцию генерации ответа\*\***:

```
```python
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch

def generate_response(prompt, model_name='qwen', max_length=50):
    # Загрузка модели и токенизатора
    model = AutoModelForCausalLM.from_pretrained(model_name)
    tokenizer = AutoTokenizer.from_pretrained(model_name)

    # Предварительная подготовка данных
    prompt = prompt.strip()
    inputs = tokenizer(prompt, return_tensors='pt')

    # Генерация ответа
    with torch.no_grad():
        outputs = model.generate(**inputs, max_length=max_length, num_return_sequences=1)

    # Преобразование ответа в текст
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)

    return response

# Пример использования
prompt = "Какой будет следующий сезон сериала 'Звездные войны'?"
response = generate_response(prompt)
```

# VLLM: openai compatible api



```
from openai import OpenAI

oclient = OpenAI(api_key='123', base_url=url)

myprompt = '''**Instruction**
Ответь на вопрос, но строго следуй формату:

<thoughts>
Твои внутренние размышления на русском (пользователь их не видит)
</thoughts>
<answer>
Ответ для пользователя
</answer>

**Question**
Сколько букв а в слове балалайка?'''
res = oclient.chat.completions.create(
    model=model_name,
    messages=[{"role": "system", "content": "You are a helpful and harmless assistant. You should think step-by-step."},
              {"role": "user", "content": myprompt}],
    temperature=0.01,
    top_p=0.9,
    max_tokens=1000,
    extra_body={
        "repetition_penalty": 1.0,
        "add_generation_prompt": True,
    }
)
if res.choices is not None:
    res = res.choices[0].message.content.strip()
print(res)
```

```
python -u -m vllm.entrypoints.openai.api_server \
--model /workdir/data/models/qwen/QwQ-32B-Preview \
--gpu-memory-utilization 0.9 \
--max_model_len 8000 \
--max_num_seqs 100 \
--port 8888
```

# VLLM: openai service: пример ответа



<thoughts>

Сколько букв "а" в слове "балалайка"? Хм, нужно посчитать количество букв "а" в этом слове. Слово "балалайка" - это название музыкального инструмента, но мне нужно сосредоточиться на счете букв.

Давай разобьем слово на отдельные буквы, чтобы легче было посчитать. Б - А - Л - А - Л - А - А - Ы - К - А. Поставлю их в ряд:

Б А Л А Л А А Ы К А

Теперь пересчитаем количество "а". Смотрим каждую букву:

1. Б - не а
2. А - это а (1)
3. Л - не а
4. А - это а (2)
5. Л - не а
6. А - это а (3)
7. А - это а (4)
8. Ы - не а
9. К - не а
10. А - это а (5)

Так, получается пять букв "а". Но лучше перепроверить, может быть, я ошиблась.

Повторяю счет:

Б А Л А Л А А Ы К А

А - 1, А - 2, А - 3, А - 4, А - 5. Да, действительно пять букв "а".

А может быть, в слове "балалайка" меньше "а"? Нет, слово именно "балалайка", с двумя "а" подряд в середине. Да, пять "а".

Уверена в ответе.

</thoughts>

<answer>

В слове "балалайка" пять букв "а".

</answer>

# VLLM: openai service: пример ответа



<thoughts>

Пользователь спрашивает, сколько букв "й" в слове "балалайка". Сначала нужно понять, как правильно написано это слово. Обычно слово "балалайка" пишется без extra "й", но возможно здесь опечатка или особенное написание. Проверим правописание слова "балалайка". Да, стандартное написание – балалайка, с одной "й". Если пользователь написал "балалаайка", то возможно, это опечатка, и ли он хочет специально вставить extra "й". Нужно уточнить, но сначала можно посчитать количество "й" в слове "балалаайка", как написано пользователем.

Слово "балалаайка" состоит из следующих букв: б, а, л, а, л, а, а, й, к, а. Давайте перечислим все "й": видим только одну "й" перед "к". Поэтому, в этом слове одна буква "й".

Но, возможно, пользователь имел в виду стандартное слово "балалайка". В этом случае, слово "балалайка" содержит две буквы "й": первая "й" между "л" и "а", вторая "й" перед "к". Но в данном случае пользователь написал "балалаайка", что может быть опечаткой.

Чтобы быть точным, нужно уточнить, но сначала можно ответить на основании того, как написано.

</thoughts>

<answer>

В слове "балалаайка", как вы его написали, буква "й" встречается один раз. Однако, если вы имели в виду стандартное слово "балалайка", то там две буквы "й". Пожалуйста, уточните, если нужно.

</answer>



# Offline Batched inference: сравнение

- Случай с длинной генерацией - огромное ускорение
- В случае генерации 1 токена, ускорения подобного нет

```
import time
prompt_token_ids = [tokenizer(d, add_special_tokens=False)['input_ids'] for d in dataset]
s = time.time()
outputs = llm.generate(prompt_token_ids=prompt_token_ids, sampling_params=sampling_params)
gen_time = time.time() - s
total_len = 0
total_tokens = 0
for output in outputs:
    generated_text = output.outputs[0].text
    total_len += len(generated_text)
    total_tokens += len(output.outputs[0].token_ids)
print(total_len / gen_time)
print(total_tokens / gen_time)
```

Processed prompts: 100% |██████████| 500/500 [00:31<00:00, 16.13it/s, est. speed input: 459.52 toks/s, output: 5921.70 toks/s]

24127.652771122684  
5912.201165616972

```
import time
from tqdm import tqdm
bs = 16

total_len = 0
s = time.time()
for i in tqdm(range(0, len(dataset[:100]), bs)):
    res = model.generate_batch(dataset[i:i+bs])
    batch_char_len = sum([len(t) for t in res[1]])
    total_len += batch_char_len
total_time = time.time() - s

print(total_len / total_time)
```

100% |██████████| 7/7 [02:14<00:00, 19.22s/it]

1259.603208250194

# Multi-gpu/Multi-node



vLLM позволяет очень легко поднять модель в multi-gpu с использованием tensor parallel или pipeline parallel, передав флаги, например:

`--tensor-parallel-size 4`

`--pipeline-parallel-size 2`

Предусмотрена возможность multi-node.

vLLM поддерживает квантизации: GPTQ, AWQ,



- Другой популярный фреймворк для инференса LLM
- Фокус не на пропускную способность, но на возможность запуска в целом:
  - Свои типы для квантизации GGUF
  - Возможность выгрузки слоев на CPU (cpu offload)
  - Множество форков и успешных “клонов” (ollama и др)

# llamacpp



```
root@e735b069dcc2:/workdir/projects/llama.cpp# CUDA_VISIBLE_DEVICES=0 ./llama-cli -m RuadapTQwen2.5-32B-instruct-GGUF/Q4_K_M.gguf -n 512 -cnv -p "You are Qwen, created by Alibaba Cloud. You are a helpful assi
```

0	NVIDIA A100-SXM4-80GB	Off	00000000:07:00.0	Off	0
N/A	26C P0	74W / 400W	7766MiB / 81920MiB	0%	Default Disabled

```
system
You are Qwen, created by Alibaba Cloud. You are a helpful assistant.
```

```
> Привет, как запустить llama-cpp cli с offload половины слоев на CPU?
```

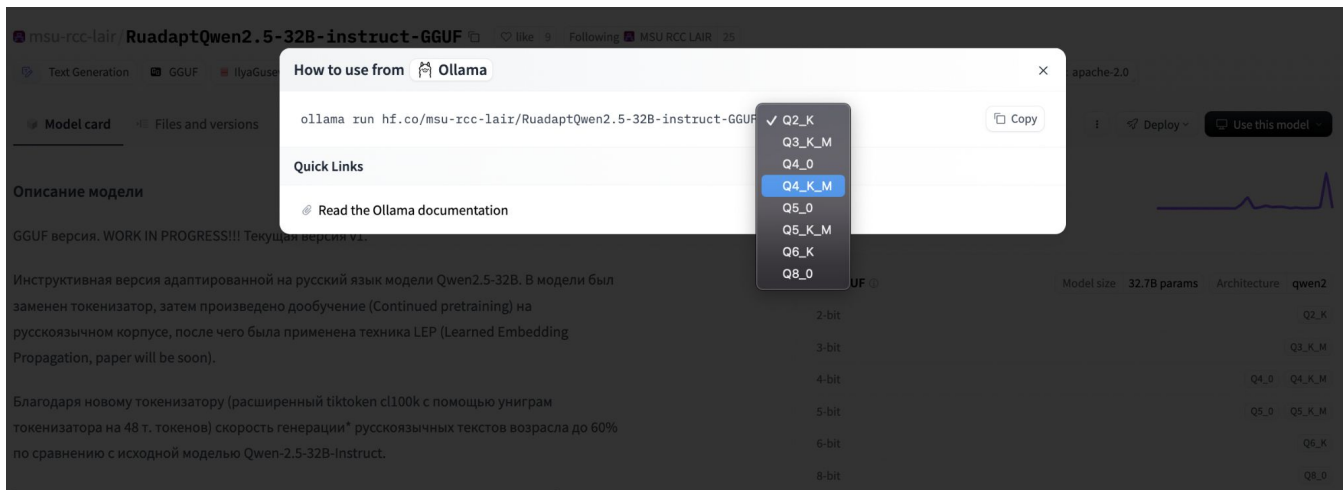
Привет! Для запуска Llama.cpp с offload (переключением нагрузки) половины слоев на CPU, нужно воспользоваться следующими параметрами при запуске CLI версии Llama.cpp. Важно помнить, что точные параметры и поддержка могут зависеть от версии Llama.cpp, которую вы используете. Вот пример команды для запуска с offloading половины слоев на CPU:

```
```bash
./main -m your_model.bin -ngl 1
```
```

В этом примере:

- `-m your_model.bin`` – указывает путь к вашему моделируемому файлу.
- `-ngl 1`` (Number of GPU Layers) – указывает количество слоев, которые остаются на GPU, при этом остальные слои будут автоматически offload'нуты на CPU.

- Удобный и популярный клон Llama.cpp
- Можно тянуть GGUF модели прямо с HF





```
>>> Чем ollama отличается от llama-cpp?
```

```
**Ollama** и **LLaMA-CPP** – это две разные реализации, но обе работают с моделями семейства LLaMA.
```

```
### Ollama:
```

- Это более новый проект.
- Разработанный как замена для `llama.cpp`, он стремится упростить работу с модели LLM (Large Language Model).
- Ориентирован на удобство и производительность: включает в себя дополнительные функции, такие как автоматическое управление моделью, возможность быстрого переключения между различными моделями LLaMA и поддержка других моделей.
- Написан на Rust и C++, что делает его относительно более быстрым и безопасным с точки зрения памяти.
- Позволяет работать через простой API или командную строку.

```
### LLaMA-CPP:
```

- Это первая известная реализация, поддерживающая моделирование модели LLaMA с использованием только CPU (без поддержки GPU).
- Написано на чистом C++ под руководством команды разработчиков от Facebook Research и сообщества open-source.
- Предоставляет низкоуровневый доступ к функциям модели, что делает его мощным для тех, кто хочет кастомизировать или интегрировать модель в свои собственные приложения напрямую.

В общем, если вы ищете более современный подход с дополнительной функциональностью и удобством использования (особенно через API), Ollama может быть предпочтительнее. Однако, для разработчиков, которым нужна полная гибкость на низкоуровневом уровне или те, кому важен полностью открытый исходный код без дополнительных абстракций, `LLaMA-CPP` всё ещё остаётся отличным выбором.

```
### Вывод:
```

- **\*\*Ollama\*\***: Простота использования, дополнительные функции и производительность.
- **\*\*LLaMA-CPP\*\***: Низкоуровневая гибкость, полное управление.

# Оценка качества LLM



Обычная оценка качества в ML: 1) Есть четкая задача 2) Есть четкий вариант правильного и не правильного ответа 3) различные формулы оценки.

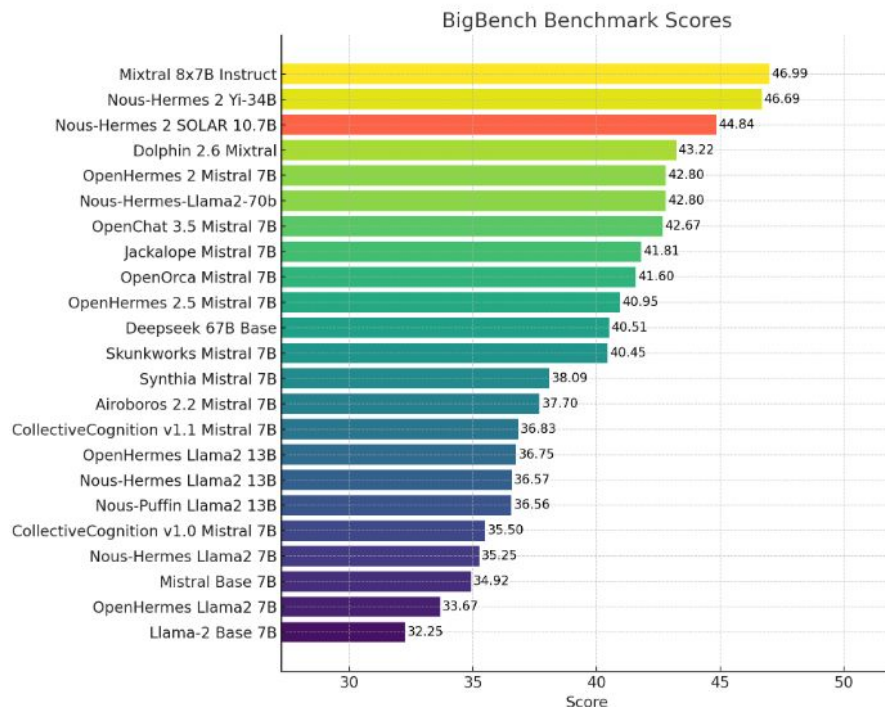
Как оценивать LLM, особенно инструктивные?

- zero-shot / few-shot при решении конкретных задач.
- Следование инструкциям.
- Полезность, вредность, токсичность.
- Сравнение side-by-side людьми
- Сравнение side-by-side намного более мощной LLM.

- Классический бенчмарк для оценки **знаний** модели.
- Вопросы с множественным выбором, 4 варианта ответа, оценка вероятности генерации A, B, C, D.
- Покрытие различных областей от физики до философии, разный уровень (колледж, старшая школа, профессиональный).
- Считают в zero-shot и few-shot с  $k=5$ .
  - Качество с few-shot у базовых и инстракт моделей обычно схожие.
- Для русского языка есть переводной вариант:  
[https://github.com/NLP-Core-Team/mmlu\\_ru](https://github.com/NLP-Core-Team/mmlu_ru)

# BigBench

- 200+ NLP задач.
- Есть Lite и Hard версии.
- Пример из карточки модели  
NousResearch/Nous-Hermes-2-SOLAR-10.7B



# Russian Super Glue



- Относительно старый бенчмарк на данный момент.
- Содержит 9 различных датасетов.
- Имеет тренировочные данные для каждого из них.
- Уровень “человека” с учетом обучения по сути уже давно превзойден.

# Russian Super Glue



## Leaderboard

We have improved the datasets. Please, change the leaderboard for the version (1.0/1.1) you are looking for, clicked on the button below.  
You can switch between the scores and inference speed leaderboard as well. Click on the button Performance.

X

Version 1.0 Performance\*

\* More information about speed scores and RAM are available here.

| Rank | Name                                                             | Team              | Link              | Score        | LIDIRus | RCB           | PARus | MuseRC        | TERRa | RUSSE | RWSD  | DaNetQA | RuCoS        |
|------|------------------------------------------------------------------|-------------------|-------------------|--------------|---------|---------------|-------|---------------|-------|-------|-------|---------|--------------|
| 1    | <a href="#">HUMAN BENCHMARK</a>                                  | AGI NLP           | <a href="#">i</a> | <b>0.811</b> | 0.626   | 0.66 / 0.702  | 0.982 | 0.806 / 0.42  | 0.92  | 0.805 | 0.84  | 0.915   | 0.93 / 0.89  |
| 2    | <a href="#">ruadapt Solar 10.7 twostage</a>                      | RCC MSU           | <a href="#">i</a> | <b>0.805</b> | 0.591   | 0.597 / 0.594 | 0.916 | 0.946 / 0.837 | 0.927 | 0.739 | 0.844 | 0.933   | 0.82 / 0.797 |
| 3    | <a href="#">Mistral 7B LoRA</a>                                  | Saiga team        | <a href="#">i</a> | <b>0.763</b> | 0.46    | 0.529 / 0.573 | 0.824 | 0.927 / 0.787 | 0.888 | 0.758 | 0.786 | 0.919   | 0.83 / 0.816 |
| 4    | <a href="#">FRED-T5 1.7B finetune</a>                            | SberDevices       | <a href="#">i</a> | <b>0.762</b> | 0.497   | 0.497 / 0.541 | 0.842 | 0.916 / 0.773 | 0.871 | 0.823 | 0.669 | 0.889   | 0.9 / 0.902  |
| 5    | <a href="#">Golden Transformer v2.0</a>                          | Avengers Ensemble | <a href="#">i</a> | <b>0.755</b> | 0.515   | 0.384 / 0.534 | 0.906 | 0.936 / 0.804 | 0.877 | 0.687 | 0.643 | 0.911   | 0.92 / 0.924 |
| 6    | <a href="#">LLaMA-2 13B LoRA</a>                                 | Saiga team        | <a href="#">i</a> | <b>0.718</b> | 0.398   | 0.489 / 0.543 | 0.784 | 0.919 / 0.761 | 0.793 | 0.74  | 0.714 | 0.907   | 0.78 / 0.76  |
| 7    | <a href="#">Saiga 13B LoRA</a>                                   | Saiga team        | <a href="#">i</a> | <b>0.712</b> | 0.436   | 0.439 / 0.5   | 0.694 | 0.896 / 0.704 | 0.865 | 0.728 | 0.714 | 0.862   | 0.85 / 0.83  |
| 8    | <a href="#">YaLM p-tune (3.9B frozen + 40K trainable params)</a> | Yandex            | <a href="#">i</a> | <b>0.711</b> | 0.364   | 0.357 / 0.479 | 0.834 | 0.892 / 0.707 | 0.841 | 0.71  | 0.669 | 0.85    | 0.92 / 0.916 |
| 9    | <a href="#">ruadapt LLaMA-2 7B LoRA</a>                          | RCC MSU           | <a href="#">i</a> | <b>0.71</b>  | 0.417   | 0.545 / 0.555 | 0.756 | 0.894 / 0.695 | 0.876 | 0.668 | 0.708 | 0.878   | 0.76 / 0.733 |
| 10   | <a href="#">FRED-T5 large finetune</a>                           | SberDevices       | <a href="#">i</a> | <b>0.706</b> | 0.389   | 0.456 / 0.546 | 0.776 | 0.887 / 0.678 | 0.801 | 0.775 | 0.669 | 0.799   | 0.87 / 0.863 |
| 11   | <a href="#">RuLeanALBERT</a>                                     | Yandex Research   | <a href="#">i</a> | <b>0.698</b> | 0.403   | 0.361 / 0.413 | 0.796 | 0.874 / 0.654 | 0.812 | 0.789 | 0.669 | 0.76    | 0.9 / 0.902  |
| 12   | <a href="#">FRED-T5 1.7B (only encoder 760M) finetune</a>        | SberDevices       | <a href="#">i</a> | <b>0.694</b> | 0.421   | 0.311 / 0.441 | 0.806 | 0.882 / 0.666 | 0.831 | 0.723 | 0.669 | 0.735   | 0.91 / 0.911 |
| 13   | <a href="#">ruT5-large finetune</a>                              | SberDevices       | <a href="#">i</a> | <b>0.686</b> | 0.32    | 0.45 / 0.532  | 0.764 | 0.855 / 0.608 | 0.775 | 0.773 | 0.669 | 0.79    | 0.86 / 0.859 |
| 14   | <a href="#">ruRoberta-large finetune</a>                         | SberDevices       | <a href="#">i</a> | <b>0.684</b> | 0.343   | 0.357 / 0.518 | 0.722 | 0.861 / 0.63  | 0.801 | 0.748 | 0.669 | 0.82    | 0.87 / 0.867 |
| 15   | <a href="#">gpt-3.5-turbo zero-shot</a>                          | Saiga team        | <a href="#">i</a> | <b>0.682</b> | 0.422   | 0.484 / 0.505 | 0.888 | 0.817 / 0.532 | 0.795 | 0.596 | 0.714 | 0.878   | 0.68 / 0.667 |
| 16   | <a href="#">Golden Transformer v1.0</a>                          | Avengers Ensemble | <a href="#">i</a> | <b>0.679</b> | 0.0     | 0.406 / 0.546 | 0.908 | 0.941 / 0.819 | 0.871 | 0.587 | 0.545 | 0.917   | 0.92 / 0.924 |
| 17   | <a href="#">xim-roberta-large (Facebook) finetune</a>            | SberDevices       | <a href="#">i</a> | <b>0.654</b> | 0.369   | 0.328 / 0.457 | 0.59  | 0.809 / 0.501 | 0.798 | 0.765 | 0.669 | 0.757   | 0.89 / 0.886 |
| 18   | <a href="#">mdeberta-v3-base (Microsoft) finetune</a>            | SberDevices       | <a href="#">i</a> | <b>0.651</b> | 0.332   | 0.27 / 0.489  | 0.716 | 0.825 / 0.531 | 0.783 | 0.727 | 0.669 | 0.708   | 0.87 / 0.868 |
| 19   | <a href="#">Saiga2 70B zero-shot</a>                             | Saiga team        | <a href="#">i</a> | <b>0.643</b> | 0.365   | 0.385 / 0.461 | 0.82  | 0.669 / 0.096 | 0.811 | 0.59  | 0.831 | 0.878   | 0.69 / 0.678 |
| 20   | <a href="#">Saiga Mistral 7B zero-shot</a>                       | Saiga team        | <a href="#">i</a> | <b>0.635</b> | 0.322   | 0.436 / 0.5   | 0.698 | 0.84 / 0.553  | 0.807 | 0.587 | 0.727 | 0.839   | 0.58 / 0.571 |
| 21   | <a href="#">ruT5-base finetune</a>                               | SberDevices       | <a href="#">i</a> | <b>0.635</b> | 0.267   | 0.423 / 0.461 | 0.636 | 0.808 / 0.475 | 0.736 | 0.707 | 0.669 | 0.769   | 0.85 / 0.847 |
| 22   | <a href="#">ruBert-large finetune</a>                            | SberDevices       | <a href="#">i</a> | <b>0.62</b>  | 0.235   | 0.356 / 0.5   | 0.656 | 0.778 / 0.436 | 0.704 | 0.707 | 0.669 | 0.773   | 0.81 / 0.805 |

- Основной разработчик бенча: Сбер (заявлено как Альянс ИИ)
- Содержит 21 задачу
- Недавно получил апдейт, обновился лидерборд

| Модель, команда                           | Результат | RWSD  | PARus | RCB           | MultiIQ       | ruWorldTree   | ruOpenBookQA  | CheGeKa       |
|-------------------------------------------|-----------|-------|-------|---------------|---------------|---------------|---------------|---------------|
| 1 GPT4o<br>MERA                           | 0.642     | 0.496 | 0.944 | 0.557 / 0.521 | 0.572 / 0.431 | 0.985 / 0.985 | 0.935 / 0.935 | 0.553 / 0.484 |
| 2 Meta-Llama-3.1-405B-In...<br>MERA       | 0.59      | 0.677 | 0.902 | 0.598 / 0.548 | 0.623 / 0.453 | 0.981 / 0.981 | 0.955 / 0.765 | 0.506 / 0.413 |
| 3 GigaChat Max<br>GIGACHAT                | 0.588     | 0.665 | 0.928 | 0.58 / 0.423  | 0.486 / 0.322 | 0.975 / 0.975 | 0.918 / 0.737 | 0.469 / 0.397 |
| 4 Mistral-Large-Instruct-2...<br>MERA     | 0.574     | 0.635 | 0.932 | 0.55 / 0.531  | 0.63 / 0.471  | 0.975 / 0.975 | 0.915 / 0.915 | 0.458 / 0.356 |
| 5 GPT4o-mini<br>MERA                      | 0.57      | 0.577 | 0.918 | 0.571 / 0.507 | 0.509 / 0.379 | 0.956 / 0.956 | 0.875 / 0.874 | 0.293 / 0.233 |
| 6 Qwen2.72B-Instruct<br>MERA              | 0.57      | 0.658 | 0.944 | 0.511 / 0.484 | 0.58 / 0.447  | 0.985 / 0.789 | 0.945 / 0.945 | 0.324 / 0.267 |
| 7 Qwen2.5-32B-Instruct-A...<br>mizdrovity | 0.567     | 0.619 | 0.93  | 0.573 / 0.539 | 0.564 / 0.426 | 0.987 / 0.987 | 0.935 / 0.935 | 0.172 / 0.12  |
| 8 Meta-Llama-3.1-70B-ins...<br>MERA       | 0.554     | 0.554 | 0.932 | 0.596 / 0.526 | 0.607 / 0.443 | 0.977 / 0.977 | 0.913 / 0.731 | 0.353 / 0.291 |
| 9 Meta-Llama-3-70B-Instr...<br>MERA       | 0.528     | 0.519 | 0.892 | 0.587 / 0.493 | 0.595 / 0.421 | 0.958 / 0.958 | 0.905 / 0.727 | 0.327 / 0.269 |
| 10 GigaChat Pro<br>GIGACHAT               | 0.516     | 0.362 | 0.884 | 0.575 / 0.258 | 0.302 / 0.212 | 0.931 / 0.931 | 0.873 / 0.7   | 0.397 / 0.329 |
| 11 Phi-3.5-MoE-instruct<br>MERA           | 0.487     | 0.465 | 0.864 | 0.546 / 0.486 | 0.446 / 0.349 | 0.971 / 0.971 | 0.88 / 0.88   | 0.178 / 0.139 |
| 12 Mixtral-8x22B-Instruct...<br>MERA      | 0.486     | 0.473 | 0.87  | 0.578 / 0.372 | 0.521 / 0.366 | 0.916 / 0.733 | 0.835 / 0.669 | 0.338 / 0.267 |
| 13 Qwen1.5-32B-Chat<br>MERA               | 0.482     | 0.377 | 0.904 | 0.521 / 0.456 | 0.452 / 0.318 | 0.931 / 0.932 | 0.86 / 0.86   | 0.119 / 0.082 |
| 14 GigaChat Lite+<br>GIGACHAT             | 0.477     | 0.435 | 0.85  | 0.53 / 0.278  | 0.287 / 0.208 | 0.886 / 0.886 | 0.785 / 0.63  | 0.268 / 0.214 |
| 15 GigaChat Lite<br>GIGACHAT              | 0.477     | 0.427 | 0.842 | 0.523 / 0.275 | 0.285 / 0.207 | 0.89 / 0.89   | 0.78 / 0.626  | 0.274 / 0.219 |
| 16 Qwen2.57B-A14B-Instruct<br>MERA        | 0.471     | 0.342 | 0.894 | 0.541 / 0.449 | 0.48 / 0.348  | 0.941 / 0.941 | 0.888 / 0.888 | 0.19 / 0.144  |
| 17 Phi-3-medium-4k-instruct<br>MERA       | 0.465     | 0.488 | 0.896 | 0.495 / 0.435 | 0.361 / 0.174 | 0.962 / 0.962 | 0.873 / 0.873 | 0.135 / 0.091 |
| 18 ruadapt llama3-8B-instr...<br>RCC MSU  | 0.447     | 0.542 | 0.828 | 0.534 / 0.446 | 0.483 / 0.334 | 0.838 / 0.837 | 0.773 / 0.772 | 0.146 / 0.101 |
| 19 Qwen2-7B-Instruct<br>MERA              | 0.445     | 0.462 | 0.82  | 0.541 / 0.353 | 0.442 / 0.333 | 0.926 / 0.742 | 0.783 / 0.782 | 0.069 / 0.043 |
| 20 Phi-3-medium-128k-inst...<br>MERA      | 0.441     | 0.485 | 0.9   | 0.546 / 0.459 | 0.361 / 0.162 | 0.954 / 0.764 | 0.863 / 0.691 | 0.134 / 0.087 |
| 21 Phi-3-small-8k-instruct<br>MERA        | 0.438     | 0.523 | 0.836 | 0.55 / 0.398  | 0.361 / 0.231 | 0.935 / 0.749 | 0.825 / 0.661 | 0.08 / 0.053  |

Spaces Vikhrmodels / small-shlepa-lb like 9 Running

App Files Community

### Small Shlepa LLM Leaderboard

LLM Benchmark Submit Analytics

Search  
Separate multiple queries with `;`

Show Rows with the Following Values

Select Columns to Display:  
 mmluproru  moviesmc  musicmc  lawmc  booksmc  model\_dtype  ppl

| model                                                   | mmluproru | moviesmc | musicmc | lawmc | booksmc | avg   |
|---------------------------------------------------------|-----------|----------|---------|-------|---------|-------|
| BlackSamorez/TechxGenus/Mistral-Large-Instruct-2487-AHQ | 0.276     | 0.715    | 0.386   | 0.623 | 0.446   | 0.473 |
| RefalMachine/zuadapt_llama3_8b_instruct_extended_led_ft | 0.225     | 0.475    | 0.311   | 0.617 | 0.392   | 0.484 |
| RefalMachine/openchat-3.5-0106                          | 0.223     | 0.454    | 0.291   | 0.624 | 0.39    | 0.396 |
| RefalMachine/IlyaGusev_saiqa_llama3_8b                  | 0.219     | 0.475    | 0.319   | 0.608 | 0.361   | 0.396 |
| google/gemma-2-9b                                       | 0.262     | 0.585    | 0.255   | 0.582 | 0.336   | 0.388 |
| apsys/vikhr-it-5.4-fp16-ozpo-v2                         | 0.188     | 0.47     | 0.281   | 0.643 | 0.334   | 0.383 |
| apsys/T-lite-instruct-0.1                               | 0.18      | 0.47     | 0.281   | 0.532 | 0.336   | 0.36  |
| Vikhrmodels/it-5.2-fp16-cp                              | 0.281     | 0.454    | 0.268   | 0.539 | 0.387   | 0.354 |
| lightblue/suzume-llama-3-8B-multilingual                | 0.194     | 0.329    | 0.26    | 0.518 | 0.297   | 0.319 |
| LakoMoor/Vikhr-2.5-7b-it                                | 0.169     | 0.329    | 0.166   | 0.319 | 0.21    | 0.238 |
| apsys/llama3_overfitted                                 | 0.124     | 0.1      | 0.077   | 0.232 | 0.087   | 0.124 |

Бенчмарк от Vikhr: “mmlu” с выбором из 12 вариантов

# Проблемы современных бенчмарков



- Качество на конкретных задачах не отражает пользовательский опыт.
- Сложно оценивать следование инструкциям.
- Модели переобучают на то, чтобы иметь более высокие значения на бенчмарках



Вместо сравнения на бенчмарках: сравнение side-by-side.

- Сравнение за счет других больших моделей.
  - MT-bench
  - alpaca-eval
- Сравнение за счет оценки людьми.
  - Lmsys

Arena (battle) Arena (side-by-side) Direct Chat Vision Direct Chat Leaderboard About Us

## LMSYS Chatbot Arena: Benchmarking LLMs in the Wild

[Blog](#) | [GitHub](#) | [Paper](#) | [Dataset](#) | [Twitter](#) | [Discord](#)

### Rules

- Ask any question to two anonymous models (e.g., ChatGPT, Claude, Llama) and vote for the better one!
- You can continue chatting until you identify a winner.
- Vote won't be counted if model identity is revealed during conversation.

### Arena Elo Leaderboard

We collect 500K+ human votes to compute an Elo-based LLM leaderboard. Find out who is the 🏆 LLM Champion!

### Chat now!

Expand to see the descriptions of 36 models

Model A

Что такое глазунья?

Глазунья — это блюдо из яиц, приготовленное путём жарки на сковороде без помешивания. В иных странах это блюдо известно как яичница-глазунья или просто жареные яйца. Яйца аккуратно разбивают так, чтобы желток остался целым, и жарят на небольшом количестве масла или сливочного масла. Жарить их можно до разной степени готовности желтка — от слегка текучего до полностью застывшего.

Глазунью часто подают на завтрак, дополняя тостами, овощами или мясными изделиями, такими как бекон или колбасы. Это очень популярное и простое в приготовлении блюдо, которое можно встретить почти в каждой кухне мира.

Model B

Что такое глазунья?

Глазунья — это блюдо, представляющее собой яйца, жаренные на сковороде без перемешивания, таким образом, чтобы желток остался целым и мягким. В некоторых странах и регионах глазунью также называют яичницей-глазуньей. При приготовлении глазуньи обычно стараются не повредить желток, чтобы он остался жидким или полужидким, а белок полностью застыл. Блюдо может подаваться с различными добавками, такими как свежие травы, сыр, овощи или колбасные изделия, в зависимости от предпочтений.

A is better B is better Tie Both are bad

Enter your prompt and press ENTER Send

New Round Regenerate Share

## 🏆 LMSYS Chatbot Arena Leaderboard

[Vote](#) | [Blog](#) | [GitHub](#) | [Paper](#) | [Dataset](#) | [Twitter](#) | [Discord](#)

LMSYS [Chatbot Arena](#) is a crowdsourced open platform for LLM evals. We've collected over **500,000** human preference votes to rank LLMs with the Elo ranking system.

Arena Elo Full Leaderboard

Total #models: **81**. Total #votes: **634676**. Last updated: April 9, 2024.

Contribute your vote 🗳️ at [chat.lmsys.org](https://chat.lmsys.org)! Find more analysis in the [notebook](#).

| Rank | Model              | Azena Elo | 95% CI | Votes | Organization | License         | Knowledge Cutoff |
|------|--------------------|-----------|--------|-------|--------------|-----------------|------------------|
| 1    | Claude-3-Opus      | 1256      | +3/-4  | 47589 | Anthropic    | Proprietary     | 2023/8           |
| 1    | GPT-4-1106-preview | 1254      | +3/-4  | 62657 | OpenAI       | Proprietary     | 2023/4           |
| 1    | GPT-4-0125-preview | 1250      | +3/-3  | 47631 | OpenAI       | Proprietary     | 2023/12          |
| 4    | Bard-(Gemini-Pro)  | 1208      | +5/-5  | 12468 | Google       | Proprietary     | Online           |
| 4    | Claude-3-Sonnet    | 1204      | +3/-3  | 57740 | Anthropic    | Proprietary     | 2023/8           |
| 6    | Command-R+         | 1194      | +5/-5  | 17404 | Cohere       | CC-BY-NC-4.0    | 2024/3           |
| 6    | GPT-4-0314         | 1189      | +4/-3  | 41292 | OpenAI       | Proprietary     | 2021/9           |
| 8    | Claude-3-Haiku     | 1182      | +3/-4  | 50689 | Anthropic    | Proprietary     | 2023/8           |
| 9    | GPT-4-0613         | 1164      | +3/-3  | 60213 | OpenAI       | Proprietary     | 2021/9           |
| 9    | Mistral-Large-2402 | 1158      | +3/-4  | 35075 | Mistral      | Proprietary     | Unknown          |
| 10   | Qwen1.5-72B-Chat   | 1153      | +4/-5  | 27050 | Alibaba      | Qianwen LICENSE | 2024/2           |
| 10   | Claude-1           | 1150      | +5/-5  | 21868 | Anthropic    | Proprietary     | Unknown          |
| 11   | Command-R          | 1149      | +3/-3  | 31871 | Cohere       | CC-BY-NC-4.0    | 2024/3           |
| 11   | Mistral-Medium     | 1148      | +4/-5  | 29807 | Mistral      | Proprietary     | Unknown          |
| 15   | Qwen1.5-32B-Chat   | 1138      | +4/-6  | 12130 | Alibaba      | Qianwen LICENSE | 2024/2           |

Note: we take the 95% confidence interval into account when determining a model's ranking. A model is ranked higher only if its lower bound of model score is higher than the upper bound of the other model's score. See Figure 3 below for visualization of the confidence intervals.

Арена (анонимно) Арена (сравнение) Таблица лидеров 10 нас

## LLM Arena: Бенчмарк LLM моделей на русском языке

llmarena.ru - ИСПОЛЬЗУЙТЕ БЕСПЛАТНО ПОСЛЕДНИЕ ВЕРСИИ ЛУЧШИХ ЧАТ-БОТОВ НА РУССКОМ

### Правила

- Задайте любой вопрос двум моделям (например, ChatGPT, T-lite, Saiga) и выберите лучший ответ!
- Продолжайте общение до тех пор, пока не выберете победителя.
- Нажмите "Очистить историю", чтобы начать новый диалог.

Выберите две модели для общения и сравнения

YandexGPT 4 Pro

RefalMachine/ruadapt\_llama3\_instruct\_lep\_saiga\_kto\_abilitirated

Разверните, чтобы увидеть описание групп моделей (всего: 31)

Модель A

Вот пример функции на Python, которая удаляет HTML-теги и пунктуацию из текста на русском и английском языках:

```
import re

def clean_text(text):
    # Удаление HTML-тегов
    text = re.sub('<.*>', '', text)

    # Замена всех знаков пунктуации на пробелы
    punctuation = '!"#$%&'\"'()*+,-./:;<=>?@[]^_`{|}~'
    for char in punctuation:
        text = text.replace(char, ' ')

    return text

# Пример использования функции
text = "Hello, world! <b>This is a test</b>"
clean_text_result = clean_text(text)
print("Исходный текст:", text)
print("Очищенный текст:", clean_text_result)
```

Эта функция сначала удаляет все HTML-теги с помощью регулярного выражения, а затем заменяет все знаки пунктуации на пробелы. Вы можете использовать эту функцию для обработки текстов на русском и английском языках.

Модель B

Вот функция на языке Python, которая удаляет HTML теги и пунктуацию из текста на русском и английском языках:

```
import re
from bs4 import BeautifulSoup

def remove_tags_and_punctuation(text):
    # Использовать BeautifulSoup для удаления HTML тегов
    soup = BeautifulSoup(text, 'html.parser')
    text = soup.get_text()

    # Использовать регулярные выражения для удаления пунктуации
    text = re.sub(r'[A-Za-z 0-9A-Za-z]*', '', text)

    return text

# Пример использования
text = '<html><body><p>Это тестовый текст на русском и английском языках.</p></body></html>'
print(remove_tags_and_punctuation(text))
```

Эта функция использует модуль BeautifulSoup для удаления HTML тегов и регулярные выражения для удаления пунктуации. В примере, текст сначала преобразуется в Unicode для корректного использования символов всех языков.

А лучше

В лучше

Ничья

Обе плохо

# Ru Arena General

- Тоже от Vikhr
- llm-as-judge
  - gpt4o
- Сравнивается против gpt-3.5

Ru Arena General

Sign in with Hugging Face

LLM Benchmark About Submit here!

Search

Separate multiple queries with `^`.

Select Columns to Display:

score  95% CI  lower  upper  avg\_tokens  std\_tokens

lc\_score

| model                                  | score | 95% CI      | avg_tokens | std_tokens | lc_score |
|----------------------------------------|-------|-------------|------------|------------|----------|
| -chatgpt-4o-latest                     | 94.74 | +0.7 / -0.9 | 693.15     | 634.2      | 56.4     |
| o1-mini                                | 93.46 | +1.0 / -1.0 | 791.18     | 647.74     | 56.22    |
| yi-lightning                           | 93.46 | +1.0 / -1.0 | 636.68     | 469.74     | 56.22    |
| claude-3-opus-20240229                 | 91.31 | +1.0 / -0.9 | 468.69     | 254.1      | 55.92    |
| gpt-4-1106-preview                     | 90.89 | +1.2 / -1.0 | 541.66     | 346.59     | 55.86    |
| o1-preview                             | 90.8  | +1.2 / -1.2 | 664.89     | 601.34     | 55.84    |
| DeepSeek, Inc.-DeepSeek-V2-Chat-0628   | 89.67 | +0.9 / -1.2 | 514.79     | 340.79     | 55.68    |
| gemini-1.5-pro-002                     | 89.07 | +1.1 / -1.3 | 639.51     | 493.3      | 55.6     |
| gemini-1.5-pro-exp-0801                | 88.88 | +1.2 / -1.3 | 547.91     | 411.58     | 55.57    |
| Qwen-Qwen2.5-72B-Instruct              | 88.25 | +1.2 / -1.3 | 557.41     | 437.32     | 55.48    |
| claude-3-5-sonnet-20240620             | 88.17 | +1.1 / -1.4 | 387.42     | 248.97     | 55.47    |
| VikhrModels-Vikhr-LLM-Testnet-01-09-24 | 87.21 | +1.0 / -1.5 | 637        | 456.77     | 55.25    |



- Известный фреймворк для оценки LLM
- На нем основаны
  - MERA
  - shlepa
  - и др. англ. бенчмарки
- Имеет ряд достоинств и некоторые недостатки (которые со временем исправляют)
  - Нельзя нормально тестировать вариант, когда часть ответа модели уже дана
  - Сложный код с большим количеством legacy



- Задача сравнения больших языковых моделей до сих пор большая проблема.
- Количество различных англоязычных бенчмарков велико (а какой лучший?), но русскоязычных почти нет.
- А что делать с конкретными предметными областями и задачами?