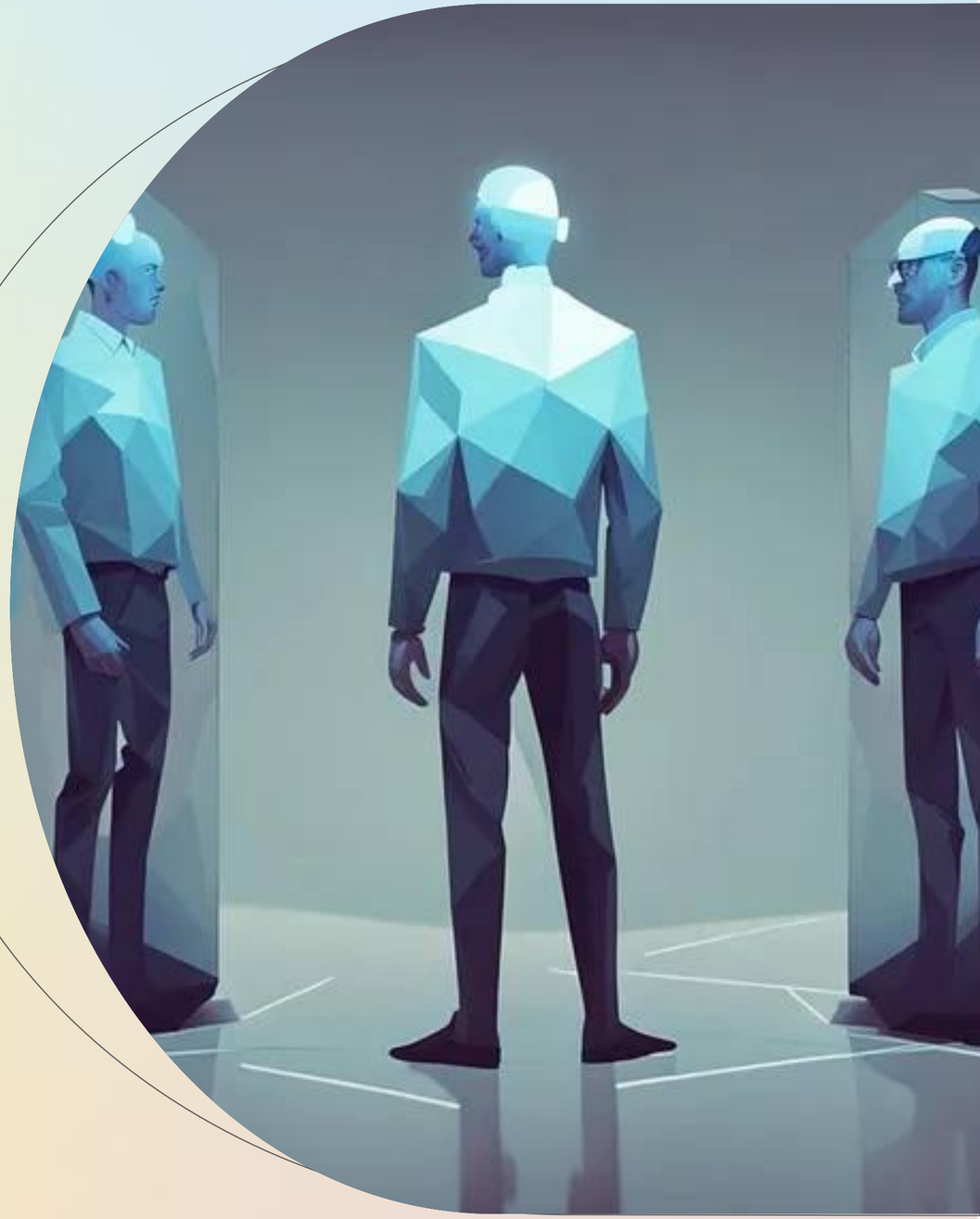
Создаем копию себя с помощью fine-tuned LLM



SAVVOV SERGEI

07 NOVEMBER 2023

HELLO THERE!

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

Основал и руководил командой разработки торговых роботов с использованием ML.

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

Саввов Сергей

sergey-savvov

@slgero

Что ждать от доклада?

1. LLM. Что за зверь?
2. Добавляем знания
3. Подготовка данных
4. Процесс обучения
5. Деплой

Что ждать от доклада?

1. **LLM. Что за зверь?**
2. Добавляем знания
3. Подготовка данных
4. Процесс обучения
5. Деплой

Что такое LLMs?

Large Language Model (LLM) – очень большая нейронная модель, которая предсказывает следующий токен на основе предсказанного ранее.

Может использоваться:

- Support бот
- Q&A по данным компании
- Автодополнение кода
- AI-tutor
- ...

Open Source Vs Closed Source

Закрытые модели:

- **OpenAI:** GPT-3.5, GPT-4
- **Anthropic:** Claude
- **Google:** BARD

Открытые модели:

- LLaMA-1, LLaMA-2
- Falcon
- Mistral AI

Вам нужно	Вы выбираете
Быстрое прототипирование	**Open Source**
Низкие расходы	**Closed Source***
Лёгкое развёртывание	**Closed Source**
Полный контроль над приложением	**Open Source**
Отсутствие внешних зависимостей	**Open Source**
Качество	**Closed Source**
Безопастность данных	**Open Source**

RAG Vs Fine-Tuning

Prompt engineering

Не добавляет новых данных в модель 🤦

Принцип работы:

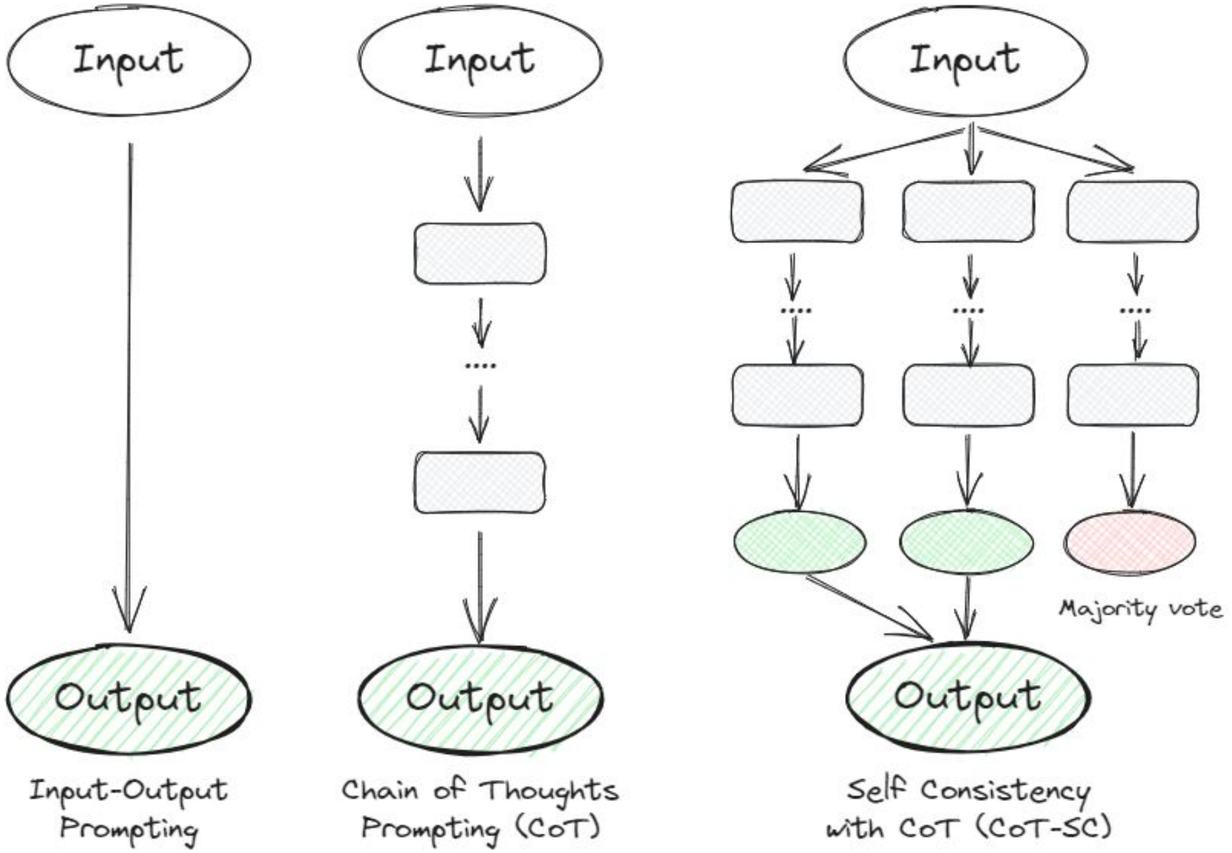
1. Экспериментируем с запросами
2. Используем “уловки”

Когда стоит использовать:

- Добиться улучшения качества
- Убрать галлюцинации

Полезные источники:

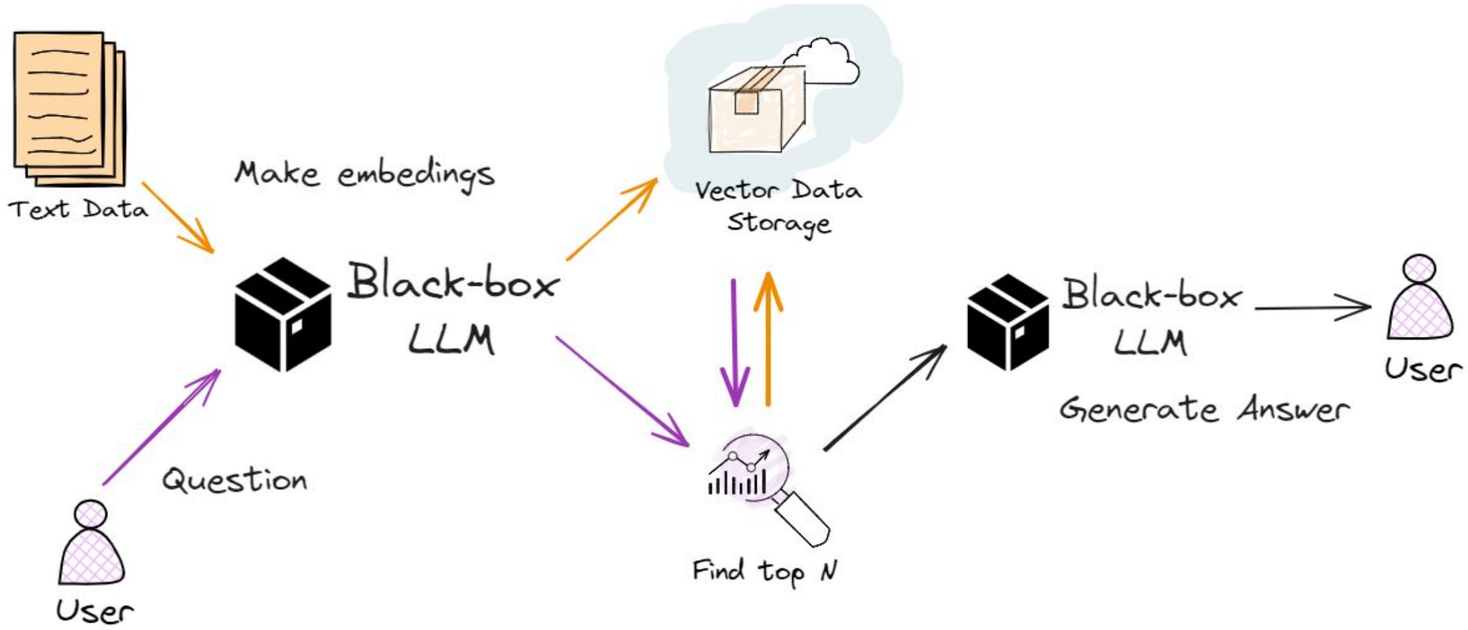
- [Prompt Engineering Guide]



Retrieval Augmented Generation

Принцип работы:

1. Текст документов → эмбеддинги
2. Вопрос пользователя → эмбеддинг
3. Ищем топ похожих
4. Генерируем ответ



Когда стоит использовать:

- Подключить модель к данным
- Строим систему вопрос-ответ

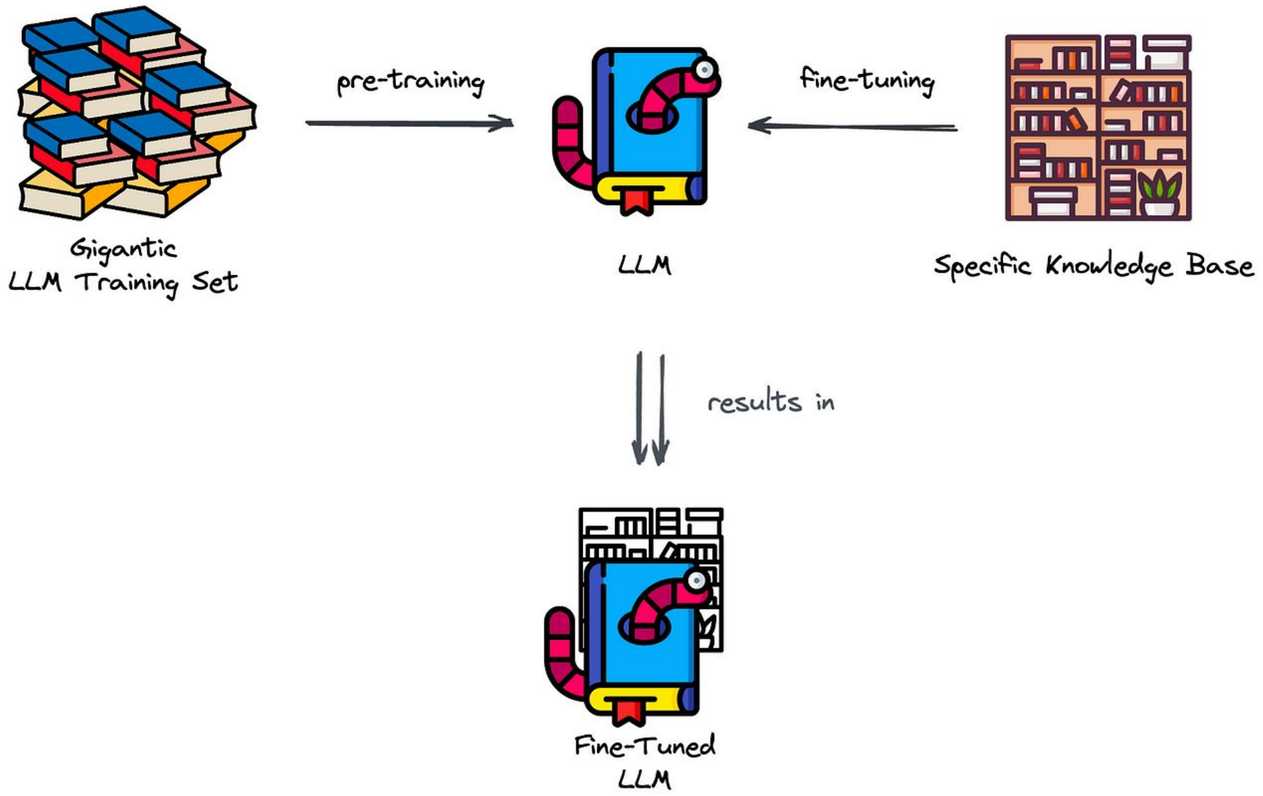
Полезные источники:

- [Building RAG-based LLM Applications for Production]

Fine-Tuning

Принцип работы:

1. Датасет с инструкциями
2. Дообучаем модель на новых данных
3. Используем новую модель



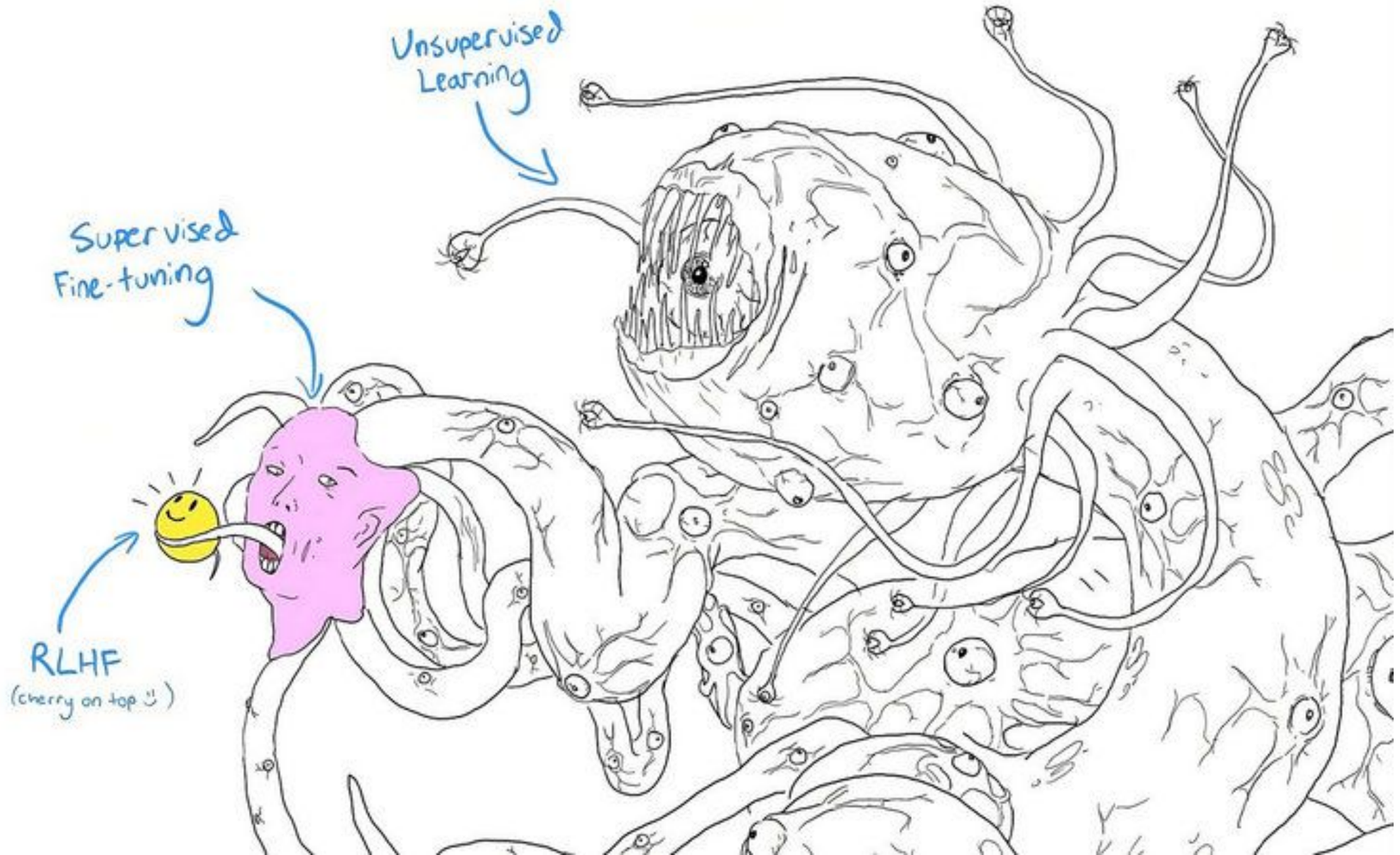
Когда стоит использовать:

- Поменять поведение модели
- Обучить доменной задаче
- Данные не часто обновляются

Полезные источники:

- [Finetuning of Large Language Models]

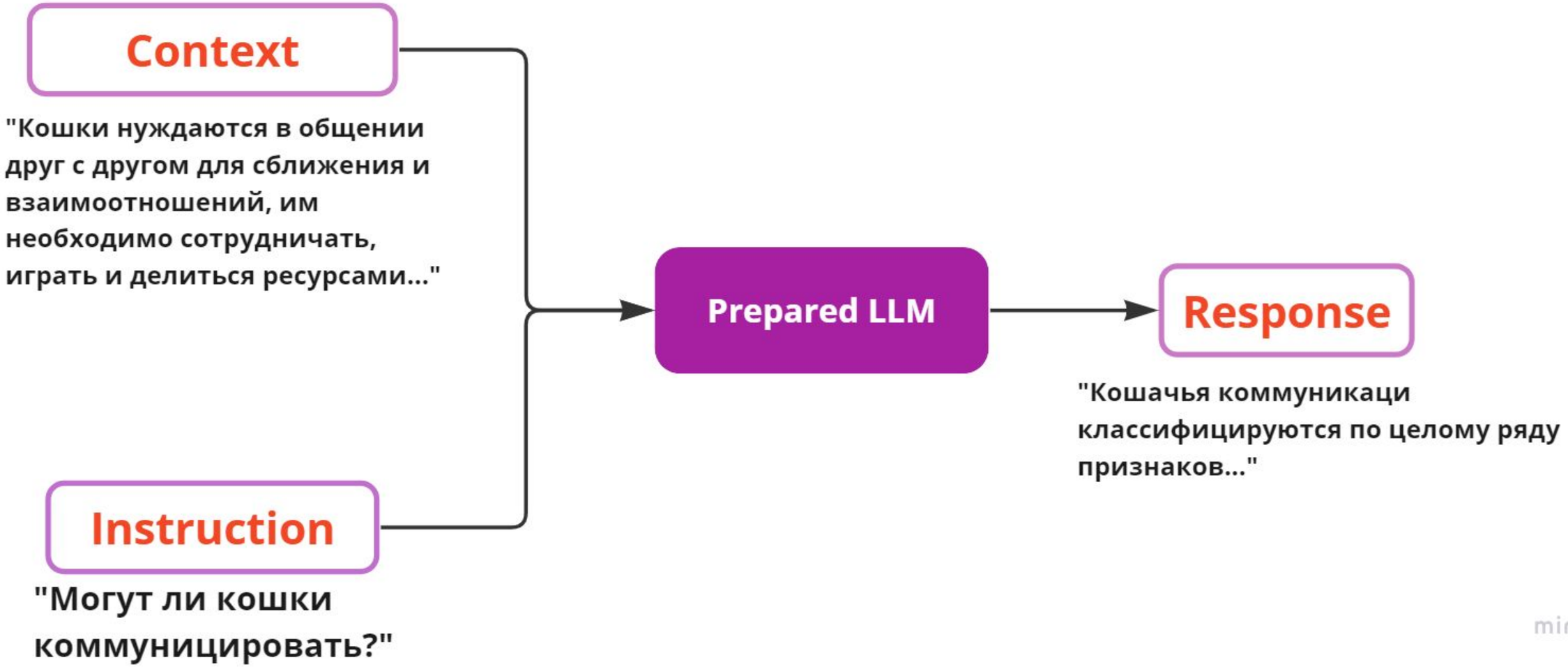
Fine-Tuning



***RLHF** – Reinforcement Learning from Human Feedback

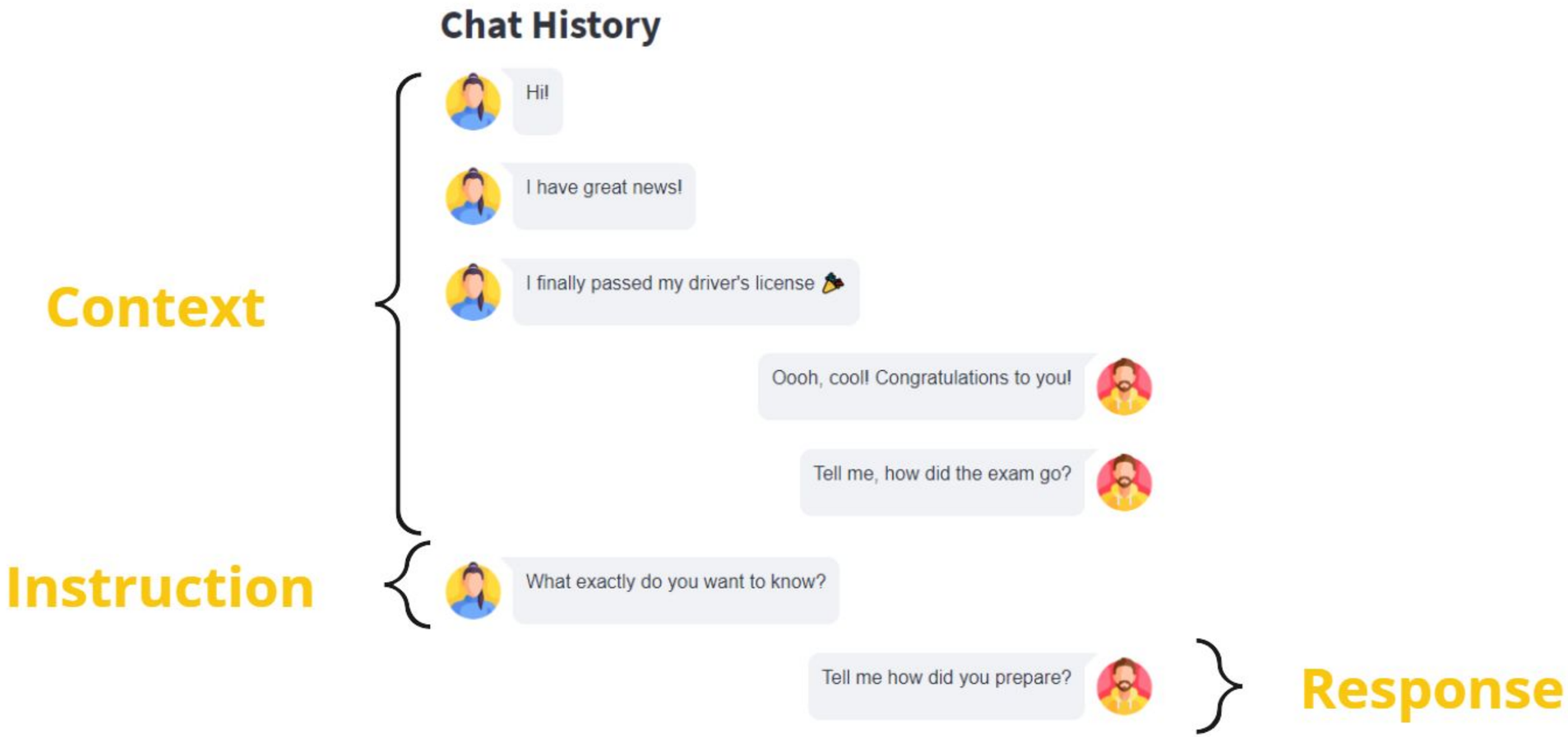
Данные это всё что вам нужно

СХЕМА ИНСТРУКЦИЙ

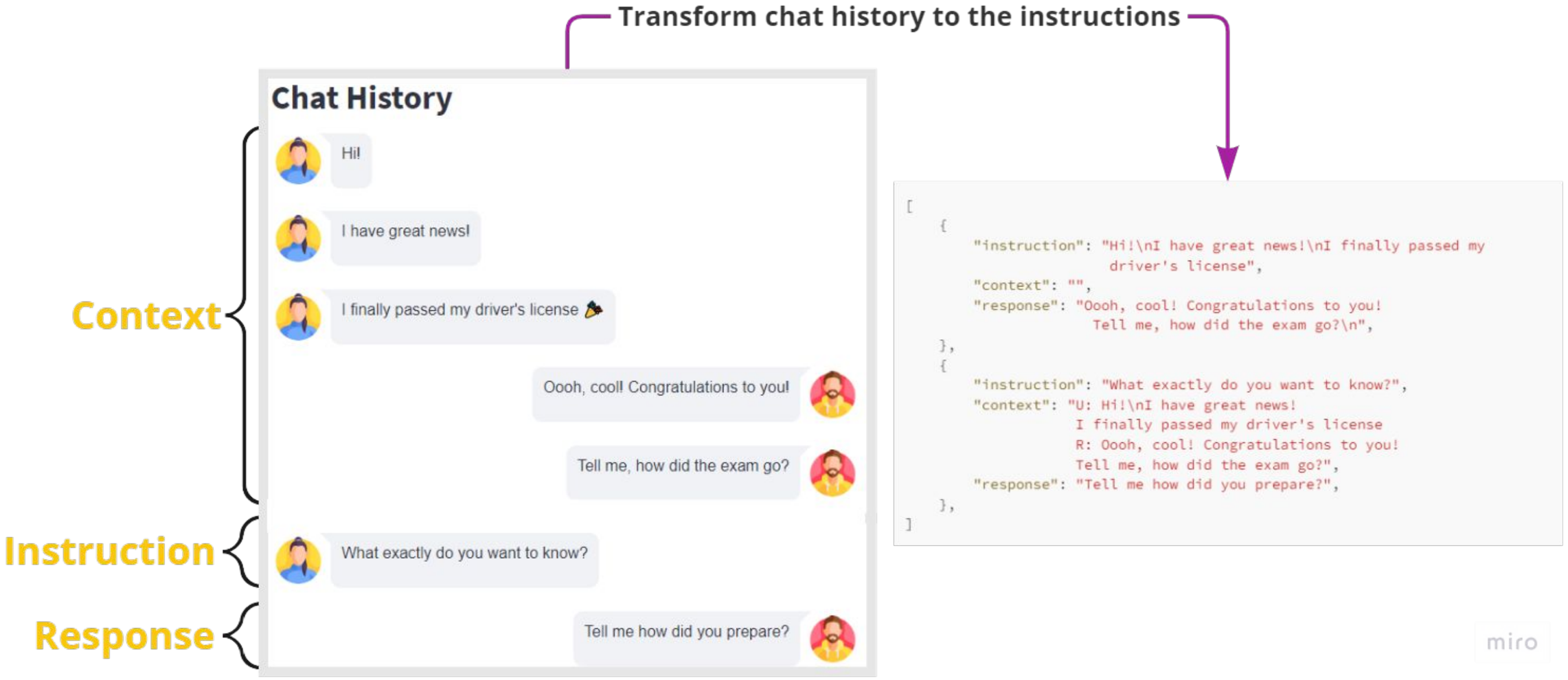


ПРИМЕР НА ИСТОРИИ ЧАТА

ПРИМЕР НА ИСТОРИИ ЧАТА



ПРИМЕР НА ИСТОРИИ ЧАТА



BEST PRACTICE

На что обращаем внимание:

- Чистота данных
- Длина контекста
- Синтетические данные
- Промпт в интрукции
- Разнообразие языка
- Чувствительная информация

Что не делаем:

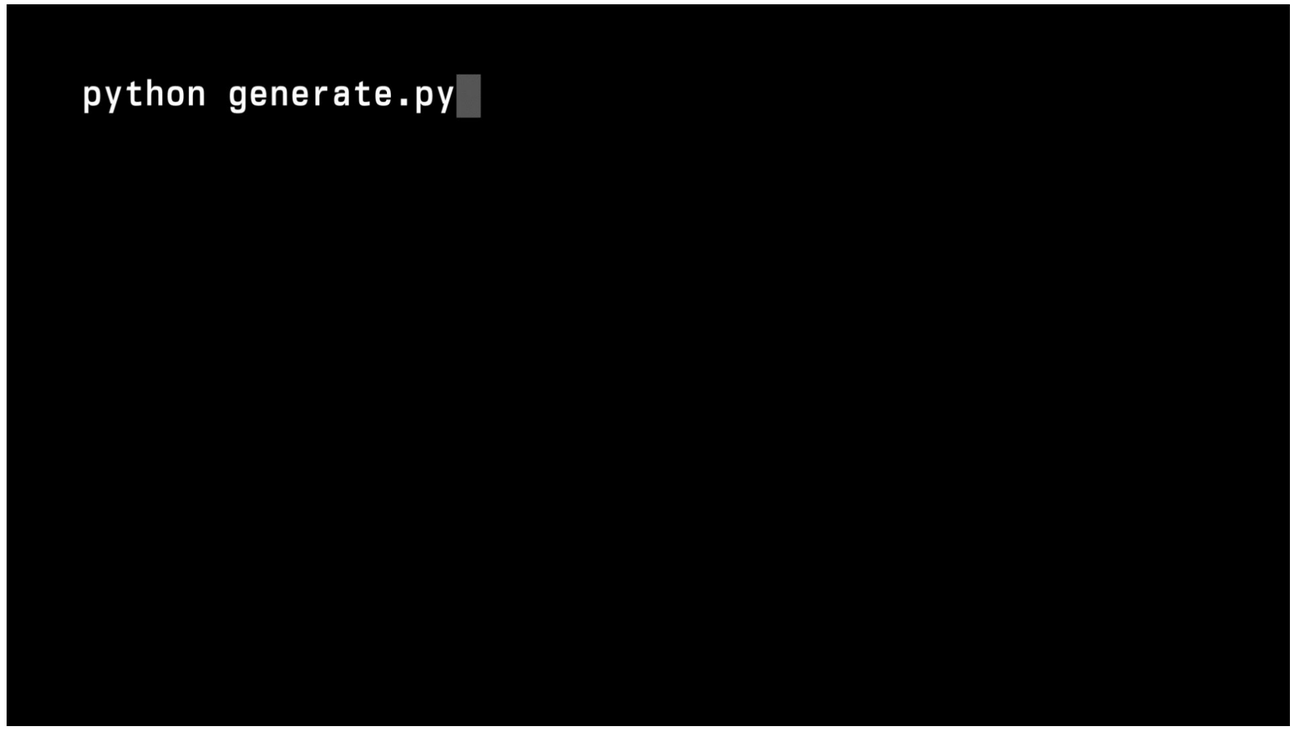
- Запихиваем всё, что есть
- Оставляем личные данные
- Пишем сложный промпт
- Готовим мало данных

Обучаем модель

БИБЛИОТЕКА ДЛЯ РАБОТЫ С LLMs

⚡ Lit-GPT

- Поддержка большинства моделей
- Можно проводить быстрые эксперименты
- Есть поддержка разных адаптеров
- Хорошо подходит для старта



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

ОБУЧЕНИЕ МОДЕЛИ ПО ШАГАМ

Что делаем:

1. Выбираем и скачиваем нужную модель
2. Подготавливаем инструкции
3. Настраиваем процесс генерации промпта
4. Запускаем обучение

```python
def generate_prompt(example: dict[str, str]) -> str:
    """Generates a standardized message to prompt the model"""
    return (
        "You (I) are chatting with a user R. Write a reply to his message"
        f"### Your previous communication:\n{example['context']}\n\n"
        f"### His new message:\n{example['instruction']}\n\n"
        f"### Your response:{example['response']}"
    )
```

```
python finetune/lora_my.py \
  --checkpoint_dir checkpoints/tiiuae/falcon-7b/ \
  --data_dir data/falcon/ \
  --out_dir out/falcon \
  --precision bf16-true
```

```
iter 1965 step 61: loss 1.5874, iter time: 261.95ms
iter 1966 step 61: loss 1.6474, iter time: 262.23ms
iter 1967 step 61: loss 1.5643, iter time: 261.93ms
iter 1968 step 61: loss 1.7458, iter time: 262.60ms
iter 1969 step 61: loss 1.4385, iter time: 261.88ms
iter 1970 step 61: loss 1.0978, iter time: 259.71ms
iter 1971 step 61: loss 1.5717, iter time: 262.13ms
iter 1972 step 61: loss 1.6004, iter time: 262.60ms
iter 1973 step 61: loss 1.7667, iter time: 263.10ms
iter 1974 step 61: loss 1.4148, iter time: 261.45ms
iter 1975 step 61: loss 1.7546, iter time: 262.17ms
```

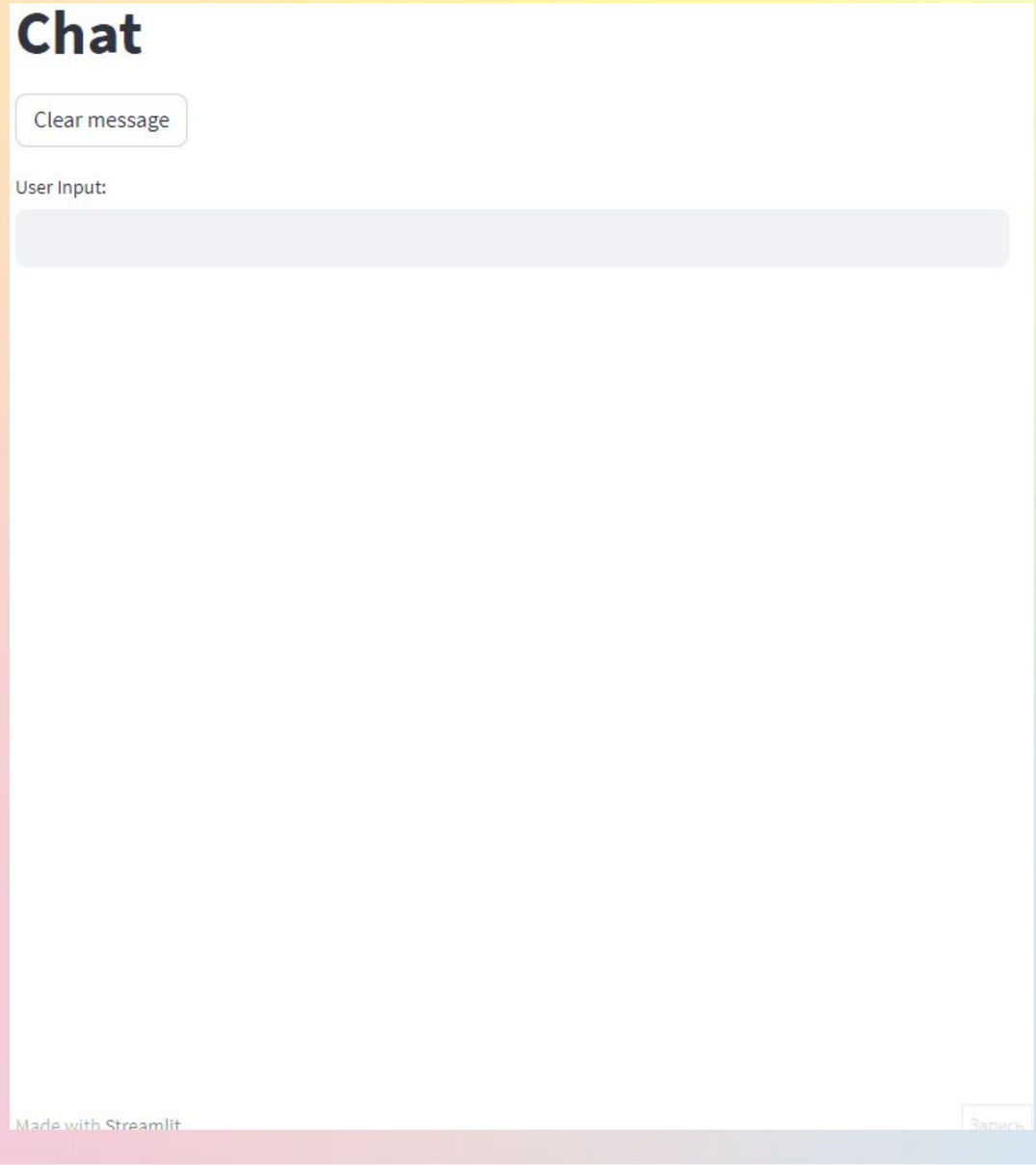
Запуск обученной модели

При запуске модели, ~~можно~~ **нужно** менять параметры:

- Квантизация
- Изменение точности весов
- Использование батч-запросов
- Распределение на несколько GPU

Можно легко запустить одной командой:

```
python generate/lora.py \
  --checkpoint_dir checkpoints/tiiuae/falcon-7b \
  --lora_path out/falcon/lit_model_lora_finetuned.pth \
  --prompt "What happened to you? Tell me" \
  --max_new_tokens 300
  --precision bf16-true
```



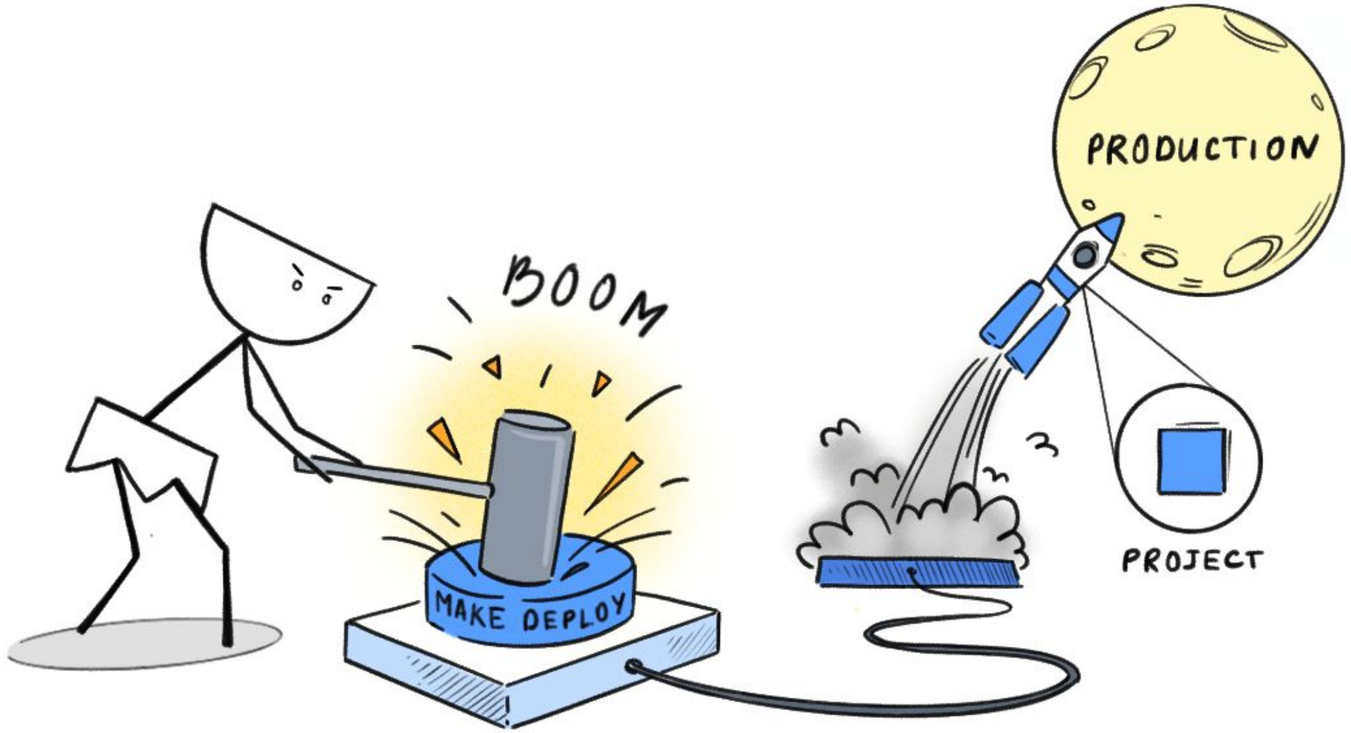
1000 и 1 фреймворк для инференса

1. LLM. Что за зверь?
2. Добавляем знания
3. Подготовка данных
4. Процесс обучения
5. **Деплой**

Зачем?

Что нам дают фреймворки:

1. Много готовых обвязок и интеграций
2. Обширное число доступных моделей
3. Наличие огромного числа оптимизаций
4. Быстрое прототипирование
5. ~~Много головной боли с установкой~~



Как выбрать?

Важна скорость:

1. Батчинг
2. Внутренние оптимизации
3. Стриминг токенов

Примеры:

- vLLM
- DeepSpeed MII

Хотим запускать на девайсах:

1. Можно запускать без GPU
2. Интеграция с Android и iOS
3. Оптимизация памяти

Примеры:

- CTranslate2
- MLC LLM

Наиболее production-ready:

1. Запуск в докере
2. Встроенный мониторинг
3. Автоскейлинг на реплики

Примеры:

- Text Generation Inference
- Ray Serve

Сравнение фреймворков

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

[Comparison of frameworks for LLMs inference]

Сравнение фреймворков

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

[Comparison of frameworks for LLMs inference]

Пример инференса LLaMA

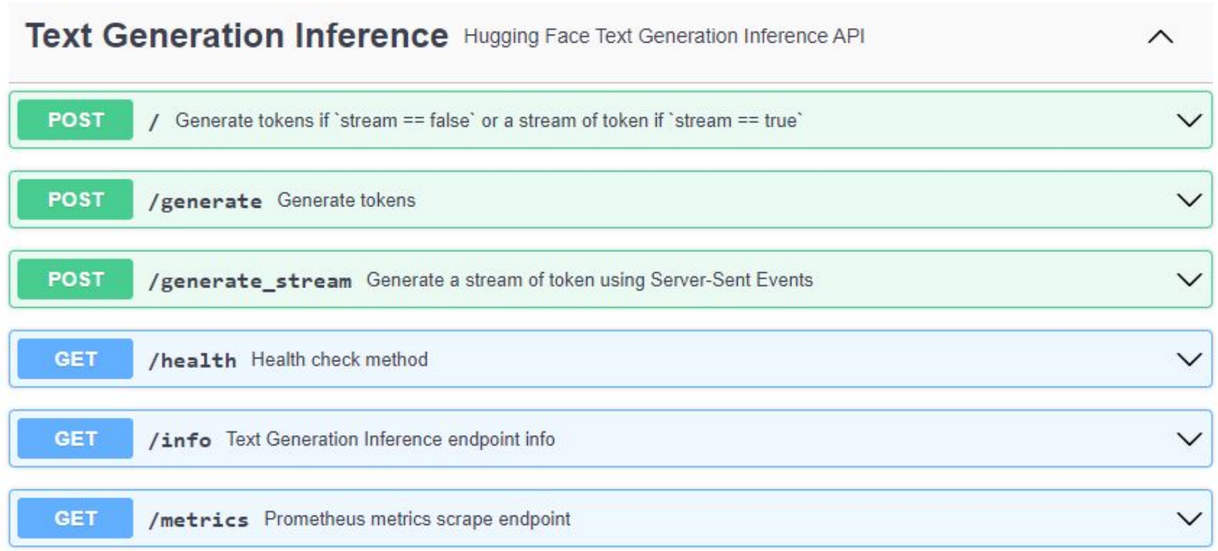
Что делаем:

1. Запускаем докер с моделью

```
mkdir data

docker run --gpus all --shm-size 1g -p 8080:80 \
  -v data:/data ghcr.io/huggingface/text-generation-inference:0.9 \
  --model-id huggyllama/llama-13b
```

2. Поднимается REST API сервер



3. Делаем запрос

```
curl 127.0.0.1:8080/generate \
    -X POST \
    -d '{"inputs":"Расскажи анекдот","parameters":{"max_new_tokens":20}}' \
    -H 'Content-Type: application/json'
```

4. Вы восхитительны 🎉

ЧТО ЧИТАТЬ ДАЛЕЕ

01. LangChain – быстрое прототипирование, построение цепочки запросов
02. Vector DataBase – хранение информации в эмбедингах
03. Fine-tuning with adapters – обучаем модель под собственную задачу
04. Prompts engineering – ваше качество сильно зависит от промпта
05. Методы уменьшения и ускорения моделей – снижаем стоимость инференса
06. Deploy libraries – улучшаем стабильность и мониторим

Спасибо за внимание!