

Своя модель: меняем манеру ответа

Лекция 1 · Практикум
Даниил Бакалин

Запускаем Qwen2.5-1.5B-Instruct, сохраняем исходный ответ и дообучаем небольшую LoRA на готовых английских диалогах в пиратском стиле. Затем повторяем независимые вопросы без инструкции о стиле.

Основной путь: Python 3.11, NVIDIA и Jupyter. Код выполняется последовательно в чистом ядре. Для копирования удобнее HTML-практикум или файл .ipynb: в PDF длинная команда установки перенесена через обратную косую черту.

Задача опыта — увидеть изменение поведения. Правильность фактов, сохранение языка и стиль оцениваются отдельно. Снижение учебной ошибки не доказывает улучшение всех ответов.

Подготовка окружения

Если Anaconda ещё нет, установите её с anaconda.com/download. Jupyter уже входит в дистрибутив. В Anaconda Prompt создайте отдельный Python и зарегистрируйте ядро:

```
conda create -n llm-course python=3.11 pip ipykernel -y
conda activate llm-course
python -m ipykernel install --user --name llm-course --display-name "LLM course"
```

Откройте JupyterLab из Anaconda Navigator. В меню Kernel → Change Kernel выберите LLM course. Создавать окружение из кода через subprocess не нужно. Интерфейс Jupyter может работать из base, а ячейки — в llm-course.

1. Установим библиотеки

%pip устанавливает пакеты в Python текущего ядра. Первая команда берёт сборку PyTorch с CUDA 12.1 из официального репозитория. Вторая устанавливает согласованный набор библиотек курса. После установки перезапустите ядро через Kernel → Restart Kernel. Установка выполняется один раз для окружения.

Версии закреплены для повторения примера. Установка меняет пакеты выбранного окружения. После неё перезапустите ядро. В терминале замените %pip на python -m pip, предварительно активировав llm-course.

```
%pip install torch==2.5.1 --index-url https://download.pytorch.org/whl/cu121
%pip install transformers==4.57.1 datasets==4.8.3 peft==0.17.1 trl==0.22.2 \
    accelerate==1.13.0 huggingface-hub==0.36.2 safetensors==0.7.0
```

2. Проверим видеокарту

torch.__version__ показывает сборку. cuda.is_available() проверяет, может ли этот Python использовать NVIDIA. assert останавливает основной путь с понятным сообщением, если CUDA недоступна. Номер 0 обозначает первую доступную видеокарту.

```
# PyTorch выполняет вычисления модели и проверяет доступ к CUDA.
import torch

print("PyTorch:", torch.__version__)
assert torch.cuda.is_available(), "CUDA недоступна: см. раздел помощи"
print("Видеокарта:", torch.cuda.get_device_name(0))
```

3. Выберем модель

Слева от / — владелец репозитория, справа — его имя. Строка совпадает с концом адреса huggingface.co/Qwen/Qwen2.5-1.5B-Instruct. Для другой модели копируют её собственный идентификатор. Здесь пока выбирается только модель.

```
# Идентификатор копируется из заголовка страницы на Hugging Face.
model_id = "Qwen/Qwen2.5-1.5B-Instruct"
```

4. Загрузим готовый генератор текста

text-generation — задача продолжения текста. model указывает репозиторий. device=0 размещает модель на первой NVIDIA. dtype="auto" использует формат весов из конфигурации; у выбранной Qwen это bfloat16, формат с двумя байтами на число. Внутренний алгоритм внимания и служебные токены выбирает библиотека.

```
# pipeline объединяет модель, токенизатор и генерацию текста.
from transformers import pipeline

chat = pipeline(
    "text-generation", model=model_id, device=0, dtype="auto"
)
```

5. Посмотрим на вход модели

Токенизатор переводит строку в идентификаторы фрагментов текста. `input_ids` — список этих чисел, `decode` выполняет обратное преобразование. Числа обозначают элементы словаря, а не оценки смысла слов.

```
# Токенизатор уже загружен вместе с моделью.
ids = chat.tokenizer("Hello, captain!")["input_ids"]
print(ids)
print(chat.tokenizer.decode(ids))
```

6. Зададим вопрос

`role="user"` обозначает пользователя, `content` содержит вопрос. Pipeline сам оформляет диалог для Qwen. `max_new_tokens` ограничивает продолжение 100 токенами, `do_sample=False` отключает случайную выборку. Из результата берём первый диалог и последнее сообщение — ответ помощника. Теперь `ask("вопрос")` вызывается одной строкой. `before` сохраняет исходный ответ.

```
# Эта функция избавляет от повторения формата диалога.
def ask(question):
    reply = chat(
        [{"role": "user", "content": question}],
        max_new_tokens=100, do_sample=False
    )
    return reply[0]["generated_text"][-1]["content"]

question = "How can I organize my study time?"
before = ask(question)
print(before)
```

7. Возьмём готовые диалоги

`load_dataset` открывает набор как таблицу. `train` — имя раздела, а не команда начала обучения. В каждой строке `messages` — реплики. Фильтр оставляет короткие вопросы, `select` берёт первые 512 подходящих строк. Ответы не переписываются, стиль не добавляется вручную.

Источник: [winglian/pirate-ultrachat-10k](https://huggingface.co/datasets/winglian/pirate-ultrachat-10k). Ответы скачиваются из репозитория без переписывания.

```
from datasets import load_dataset

# В наборе уже есть вопросы и ответы в пиратском стиле.
data = load_dataset("winglian/pirate-ultrachat-10k", split="train")

# Возьмём 512 примеров с вопросами короче 300 символов.
data = data.filter(lambda row: len(row["messages"][0]["content"]) < 300)
data = data.select(range(512))
print(data[0])
```

8. Добавим обучаемую часть

PEFT — библиотека экономного дообучения. LoRA добавляет небольшие обучаемые матрицы к преобразованиям модели. CAUSAL_LM обозначает предсказание следующего токена по предыдущим. Стандартная конфигурация PEFT для Qwen выбирает поддерживаемые слои; базовые веса остаются замороженными.

```
from peft import LoraConfig

# Для первого опыта используем стандартные настройки LoRA.
adapter = LoraConfig(task_type="CAUSAL_LM")
```

9. Зададим размер эксперимента

SFT — обучение на примерах желаемого поведения. Здесь: 400 обновлений; learning_rate=0.0003 задаёт масштаб изменения; по 2 диалога обрабатываются за проход, 4 прохода накапливаются перед обновлением. Это 8 примеров на шаг. max_length ограничивает диалог 384 токенами, длинный хвост обрезается. output_dir — папка результатов, report_to="none" отключает внешние сервисы журналирования. Остальные параметры берутся из выбранной версии TRL.

```
from trl import SFTConfig, SFTTrainer

settings = SFTConfig(
    output_dir="pirate-output",
    max_steps=400, learning_rate=3e-4,
    per_device_train_batch_size=2, gradient_accumulation_steps=4,
    max_length=384,
    report_to="none"
)
```

10. Запустим дообучение

Передаём загруженную модель, её токенизатор, диалоги, LoRA и настройки. SFTTrainer готовит последовательности, собирает порции данных, вычисляет ошибку и обновляет параметры адаптера. train() запускает цикл. Прогресс появляется в выводе ячейки; loss — ошибка предсказания токенов, а не доля правильных ответов.

```
trainer = SFTTrainer(
    model=chat.model, processing_class=chat.tokenizer,
    train_dataset=data, peft_config=adapter, args=settings
)
trainer.train()
```

11. Посмотрим на ход обучения

В `log_history` есть измерения `loss` и итоговая статистика. Берём записи с `loss` и печатаем шаг и усреднённую ошибку. График в лекции строится по такому журналу реального запуска; учебная анимация показана отдельно.

```
# Журнал доступен без подключения внешнего сервиса.
for row in trainer.state.log_history:
    if "loss" in row:
        print(row["step"], round(row["loss"], 3))
```

12. Повторим тот же вопрос

`trainer.model` содержит базу с адаптером. `eval()` включает режим использования. Вопрос и настройки генерации не меняются, инструкции «говори как пират» нет. Сравниваем сохранённый `before` с `after`.

```
# Подключаем обученный адаптер к тому же генератору.
chat.model = trainer.model
chat.model.eval()

after = ask(question)
print("ДО:\n", before)
print("ПОСЛЕ:\n", after)
```

13. Проверим другой вопрос

Первый вопрос проверяет перенос стиля на другую тему, второй — на русский язык. Английский набор не гарантирует такой перенос. Смотреть нужно и на стиль, и на смысл: пиратское приветствие само по себе не делает ответ хорошим.

```
print(ask("Explain what a neural network is in two sentences."))
print(ask("Скажи, как лучше учиться программированию?"))
```

14. Сохраним результат

В `pirate-adapter` записываются веса LoRA, конфигурация и токенизатор. Базовая модель не копируется: при следующем запуске нужны и база, и адаптер. База уже находится в локальном кэше. Отдельный блокнот показывает загрузку в новом сеансе.

```
trainer.save_model("pirate-adapter")
chat.tokenizer.save_pretrained("pirate-adapter")
```

15. Переключим сохранённую LoRA

`load_adapter` загружает вторую LoRA с именем `long` к той же базовой Qwen. `set_adapter` выбирает одну добавку: веса двух адаптеров здесь не складываются. `default` — имя первой LoRA, созданной `SFTTrainer`. `eval()` включает режим ответа. Архив длинного адаптера доступен в материалах, его папку нужно разместить рядом с блокнотом.

Дополнительная демонстрация: скачайте архив адаптеров из материалов курса и разместите `pirate-adapter-long` рядом с блокнотом. Длинная версия продолжает первые 400 шагов ещё на 800 при `learning_rate=0.0001`. Эта ячейка не запускает повторное обучение.

```
# Сначала скачайте и распакуйте длинный адаптер рядом с блокнотом.
chat.model.load_adapter("pirate-adapter-long", adapter_name="long")
chat.model.set_adapter("long")
chat.model.eval()
print(ask(question))
```

```
# Вернём адаптер, обученный в этом сеансе.
chat.model.set_adapter("default")
```

Самостоятельный эксперимент

Возьмите другую модель, данные или цель дообучения. Измените стиль, формат ответа, тематическое поведение или исследуйте перенос на другой язык. Подготовьте контрольные вопросы до обучения, сохраните ответы до и после. Разберите удачные и неудачные случаи.

Использовать любые инструменты разрешено. Пришлите через форму ссылку на GitHub или архив с кодом, версиями, замыслом, режимом обучения и результатами. Ссылка на форму будет предоставлена отдельно. Не включайте токены доступа.

Другой компьютер

При нехватке видеопамати начните с `Qwen2.5-0.5B-Instruct`, `per_device_train_batch_size=1` и `max_length=256`. Перезапустите ядро, чтобы освободить старые модели. Меньшая модель и другая длина меняют условия опыта.

Для CPU используйте соответствующую сборку PyTorch, `device=-1` и `dtype=torch.float32` в `pipeline`; `use_cpu=True` и `bf16=False` в `SFTConfig`. Первые `max_steps=10` служат только проверкой запуска. Время GPU не является прогнозом времени CPU. Подробная помощь находится в `docs/troubleshooting.html`.

Источники и материалы

Модель: [Qwen2.5-1.5B-Instruct](#).

Обучение: [TRL 0.22.2](#), [SFTTrainer](#).

LoRA: [Hu et al., 2021](#).