Министерство науки и высшего образования Российской Федерации

Национальный исследовательский университет ИТМО

Инфраструктура больших данных

Весна

2025

Лабораторная работа №1

**КЛАССИЧЕСКИЙ ЖИЗНЕННЫЙ ЦИКЛ РАЗРАБОТКИ МОДЕЛЕЙ**

**МАШИННОГО ОБУЧЕНИЯ**

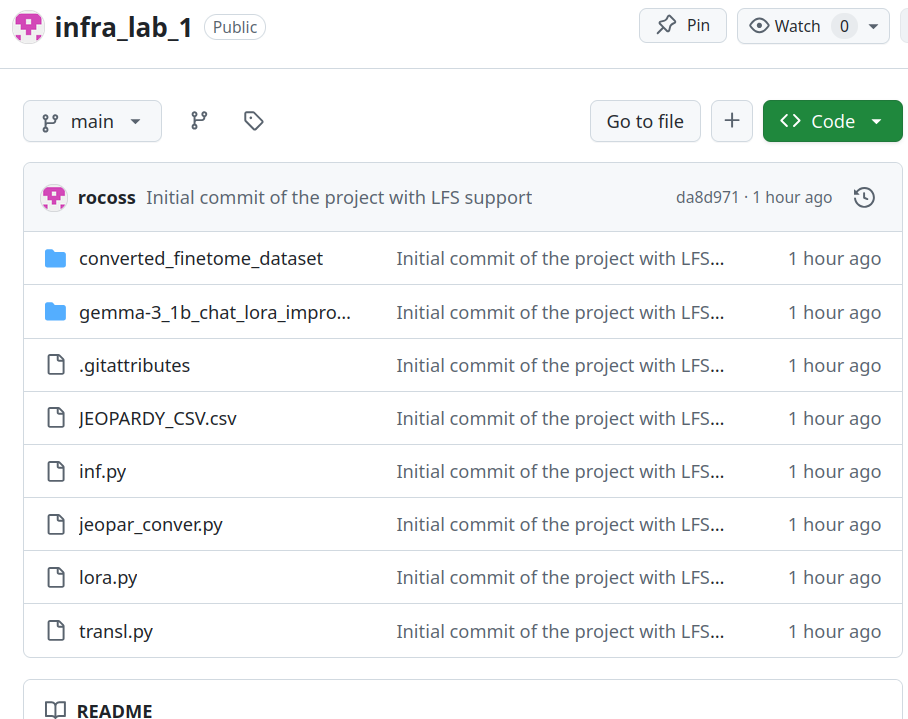
Цель работы:

Получить навыки разработки CI/CD pipeline для ML моделей с достижением метрик моделей и качества.

Ход работы:

1. Создать репозитории модели на GitHub, регулярно проводить commit +push в ветку разработки, важна история коммитов;

Для выполнения лабраторной работы был создан репозиторий <https://github.com/rocoss/infra_lab_1>



В него была загружена модель и код с помощью которого можно было ее дообучать и выполнять инференс.

2. Провести подготовку данных для набора данных, согласно варианту задания;

По варианту работы мне достался набор данных

https://www.kaggle.com/tunguz/200000-jeopardy-questions

Набор содержит 216 930 записей вопросов и ответов из американской телевикторины "Jeopardy!", каждая из которых включает подробную информацию о вопросе.

Каждая запись в наборе данных включает следующие поля:

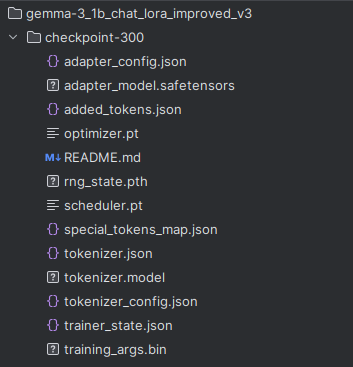
* **category**: Категория вопроса (например, "HISTORY").
* **value**: Денежная стоимость вопроса в долларах (например, "$200"). Для вопросов из раундов "Final Jeopardy!" и "Tiebreaker" это значение отсутствует.
* **question**: Текст вопроса (подсказки).
* **answer**: Текст правильного ответа.
* **round**: Раунд игры, в котором был задан вопрос ("Jeopardy!", "Double Jeopardy!", "Final Jeopardy!" или "Tiebreaker").
* **show\_number**: Порядковый номер выпуска шоу.
* **air\_date**: Дата выхода выпуска в эфир в формате ГГГГ-ММ-ДД.

Моей задачей было обучить модель отвечать на вопросы исходя из вопросов викторины, для этого необходимо было конвертировать датасет в формат пригодный для работы. Для это был написан скрипт jeopar\_conver.py, который переводит файл JEOPARDY\_CSV.csv в формат набора данных FineTome-100k, известный как ShareGPT. Этот формат является стандартом для наборов данных, предназначенных для обучения и дообучения (fine-tuning) инструктивных и диалоговых языковых моделей. Его основная задача — структурировать взаимодействие между пользователем и моделью в виде беседы. Каждый ход в диалоге — это словарь с двумя обязательными ключами: "from": Указывает, кто является автором сообщения. Обычно используются значения "human" (человек, пользователь) и "gpt" (модель, ассистент). "value": Содержит непосредственно текст сообщения.

Также дополнительно были реализованы необходимые механизмы обработки данных. Для защиты от дубликатов скрипт сначала выполняет внутреннюю очистку новых данных, а затем проводит перекрестную проверку с существующим набором, чтобы гарантировать добавление только уникальных записей. Дополнительно была проведена очистка от html тегов (например, <a href=""http://www.j-archive.com/media/2004-12-31\_DJ\_26.mp3"">Ripped from today's headlines, he was a turtle king gone mad; Mack was the one good turtle who'd bring him down</a>"), для этого использовались регулярные выражения, потому что текст внутри тега нужен как вопрос в датасете.

3. Разработать ML модель с ЛЮБЫМ классическим алгоритмом классификации, кластеризации, регрессии и т. д.;

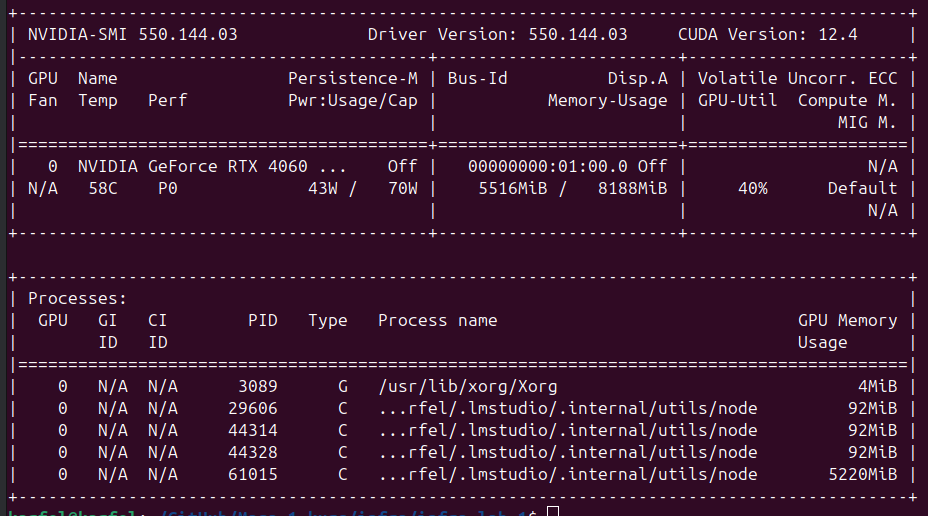
Так как задачей было обучить языковую модель было принято решение воспользоваться надежным методов peft предназначенным для дообучения инструктивных и диалоговых языковых моделей с помощью библиотеки unsloth по методу lora.



За базовую модель для дообучения была выбрана gemma-3 на 1b параметров.

4. Конвертировать модель из \*.ipynb в .py скрипты, реализовать API сервис с методом на вывод модели, фронтальная часть по желанию;

Обучение проводилось на моем локальном ноутбуке с привлечением видеокарты, все коды для обучения добавлены в репозиторий.



5. Покрыть код тестами, используя любой фреймворк/библиотеку;

Были проведены модульные тесты, которые проверяют корректность логики и последовательности шагов в пайплайне обучения модели (`train.py`), не запуская при этом реальное, ресурсоемкое обучение. Если кратко, тесты подтвердили, что основной скрипт:

1. Правильно вызывает загрузку модели: Проверено, что функция загрузки модели (`FastLanguageModel.from\_pretrained`) вызывается один раз с верными параметрами (имя модели, длина последовательности, 4-битная загрузка).

2. Корректно настраивает LoRA: Удостоверились, что к модели применяются PEFT-настройки для LoRA.

3. Верно конфигурирует тренер (`SFTTrainer`):

Протестировано, что тренер инициализируется с правильными параметрами, такими как:

* Скорость обучения.
* Размер батча.
* Путь для сохранения результатов.
* Максимальная длина последовательности.

4. Корректно обрабатывает данные:

Проверено, что датасет правильно загружается и корректно разделяется на обучающую и валидационную выборки в соотношении 90/10.

5. Соблюдает последовательность действий:

Тест гарантирует, что после всех настроек вызываются ключевые методы — сначала `train()` (обучение), а затем `save\_pretrained()` (сохранение модели и токенизатора).

Тестирование проходило методом мокирования (mocking), т.е когда все тяжелые операции (загрузка моделей с Hugging Face, вычисления на GPU, работа с файлами) были заменены на "заглушки" (`MagicMock`), которые лишь имитируют их поведение. Это позволило быстро и изолированно проверить логику, не требуя GPU и не скачивая гигабайты данных.

6. Задействовать DVC;

7 Использовать Docker для создания docker image.

8 Наполнить дистрибутив конфигурационными файлами:

• config.ini: гиперпараметры модели;

• Dockerfile и docker-compose.yml:

конфигурация

создания

контейнера и образа модели;

• requirements.txt: используемые зависимости (библиотеки) и их

версии;

• dev\_sec\_ops.yml: подписи docker образа, хэш последних 5

коммитов в репозитории модели, степень покрытия тестами

(необязательно);

• scenario.json: сценарии тестирования запущенного контейнера

модели (необязательно).

9 Создать CI pipeline (Jenkins, Team City, Circle CI и др.) для сборки docker

image и отправки его на DockerHub, сборка должна автоматически

стартовать по pull request в основную ветку репозитория модели;

10 Создать

CD

pipeline

для

запуска

контейнера

и

проведения

функционального тестирования по сценарию, запуск должен стартовать

по требованию или расписанию или как вызов с последнего этапа CI

pipeline;

11 Результаты функционального тестирования и скрипты конфигурации

CI/CD pipeline приложить к отчёту.

Результаты работы:

1 Отчёт о проделанной работе;

2 Ссылка на репозиторий GitHub;

3 Ссылка на docker image в DockerHub;

4 Актуальный дистрибутив модели в zip архиве.

Обязательно обернуть модель в контейнер (этап CI) и запустить тесты

внутри контейнера (этап CD).