Сравнение полного и параметро-эффективного дообучения трансформеров для многоязычного анализа текстов

Подготовила студентка **ВГУ, ФКН,**
3-го курса 6-й группы Полина Шапиро

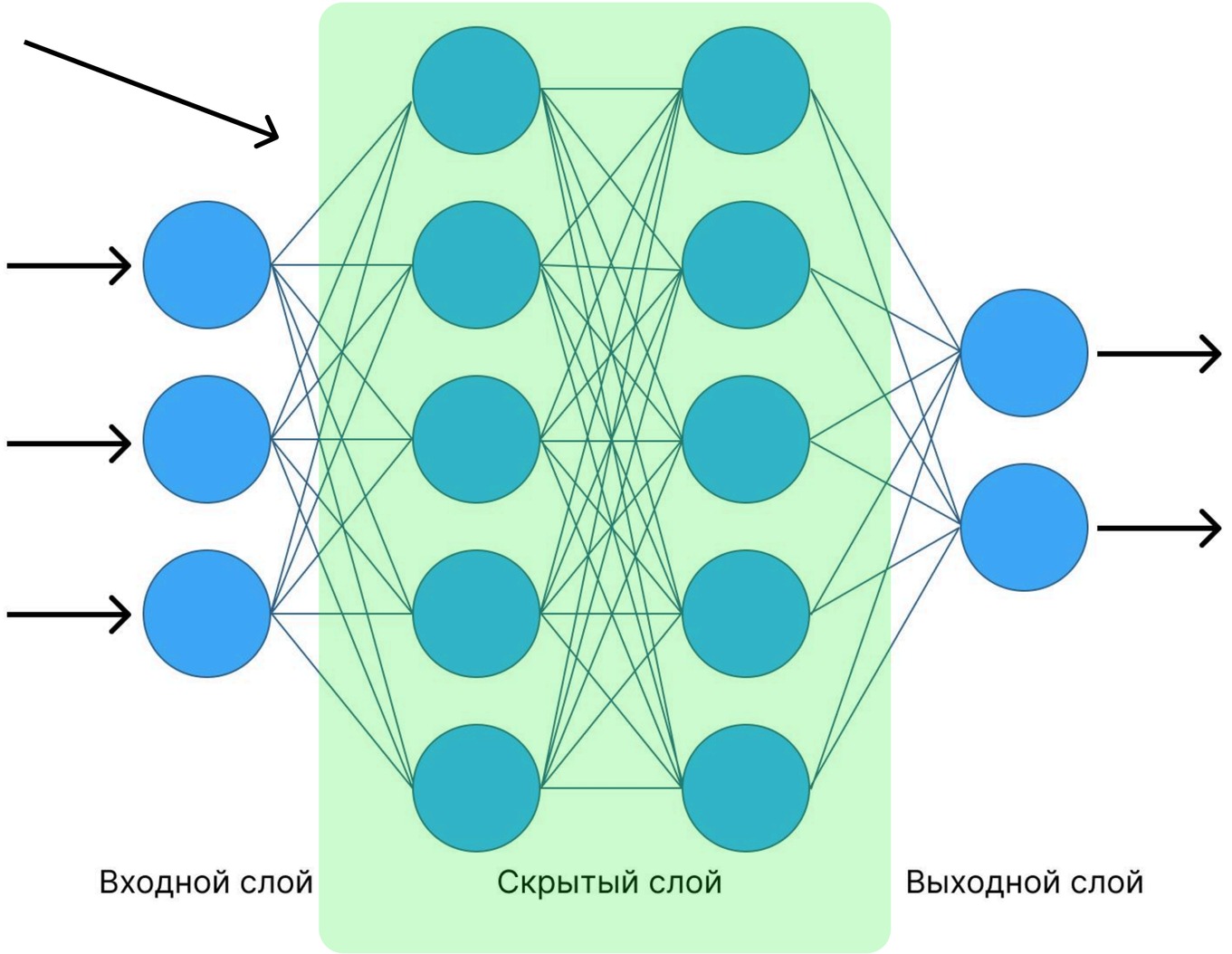
09.06.2025

01 Актуальность

Языковые модели показывают высокую точность, но за это приходится платить огромным числом параметров. Практические задачи почти всегда требуют дообучения моделей, и из-за их размера растут и вычислительные затраты

Скрытый слой H + веса W на входе, между ним и на выходе

Схема работы нейросети



01 Full fine-tuning

Используется классический способ адаптации: размораживаются все параметры предобученной модели и заново обучаем их на целевом датасете. Такой подход даёт максимальное качество, но требует хранить и обновлять сотни миллионов весов, поэтому занимает много видеопамяти и долго считается при каждой новой задаче.

02 Альтернатива — PEFT

Существует иной способ дообучения — Parameter-Efficient Fine-Tuning (PEFT). Вместо того чтобы перекраивать все сотни миллионов весов, мы «замораживаем» предобученную модель и добавляем к ней крошечные адаптеры — доли процента от исходного размера. Обучаются только эти лёгкие вставки, поэтому память, время и энергия расходуются в разы меньше, а основное знание модели сохраняется.

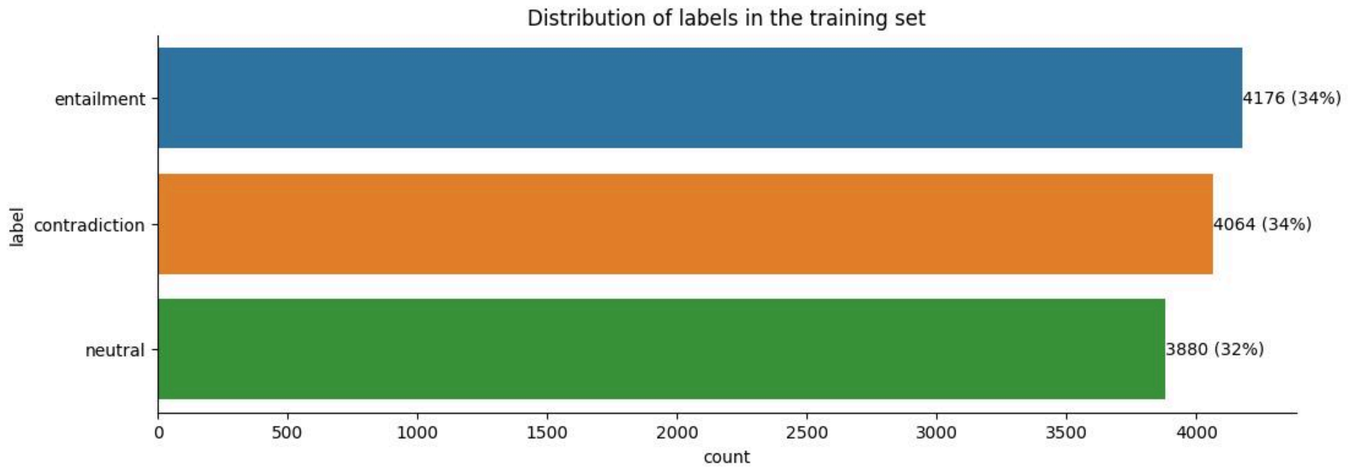
Исследование

В этой работе я исследую параметро-эффективное дообучение (PEFT) на примере метода LoRA и сравниваю его с полным fine-tuning. Сравнение проводится по времени обучения и потреблению ресурсов, а в завершение я демонстрирую веб-интерфейс, позволяющий «потрогать» полученную модель вживую

Тестовые данные

В качестве тренировочных данных был выбран публичный челлендж с Kaggle — “**Contradictory, My Dear Watson” .**

Для каждой пары предложений (premise, hypothesis) требуется определить, логически следует ли гипотеза из посылки (entailment), не связана ли она с ней (neutral) или противоречит ей (contradiction).

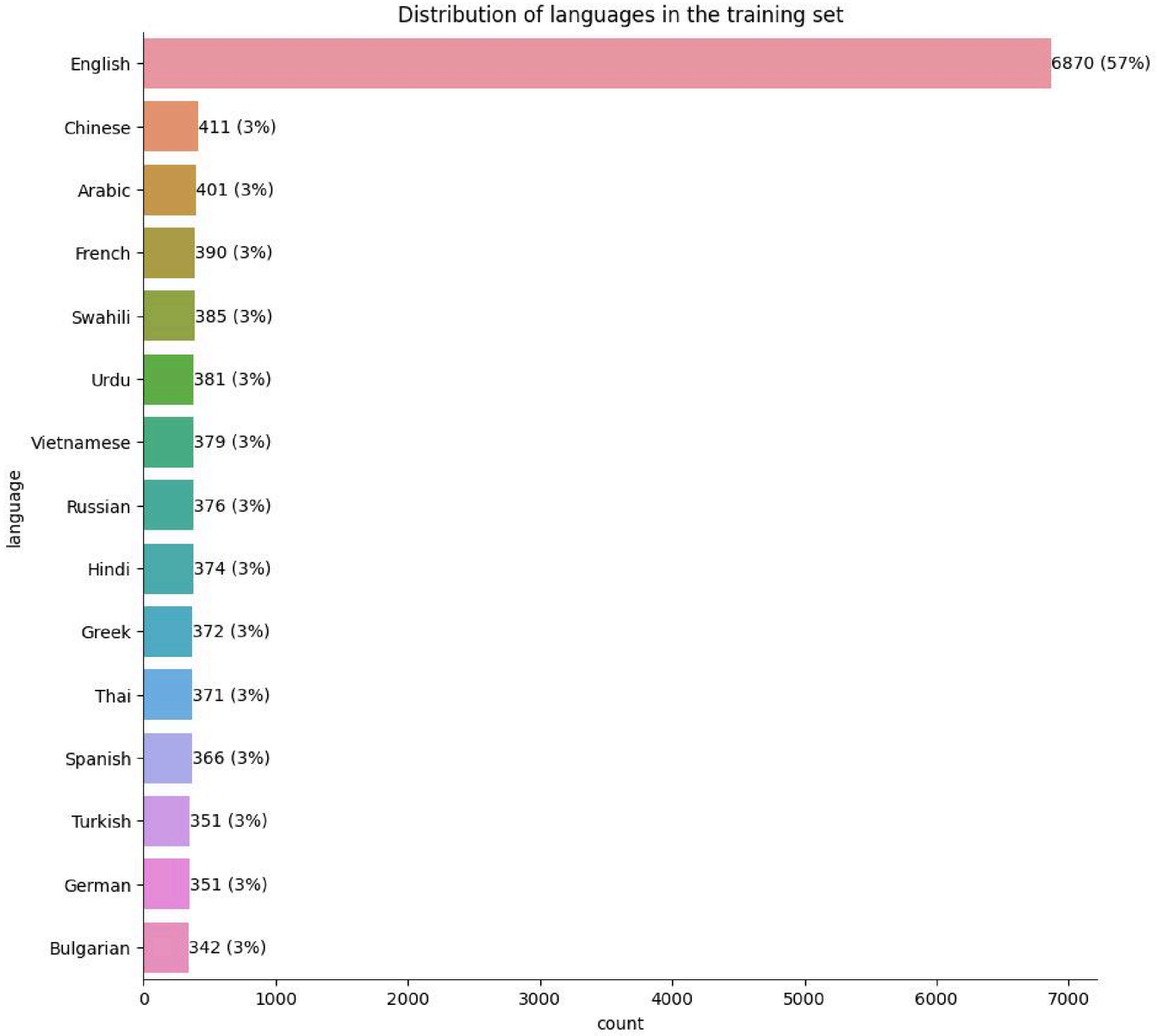


Статистика по сету

→ Корпус включает ~12 100 обучающих и ~5 200 тестовых пар на **15 языках**.

→ Английский формирует **~ 57 %** примеров

→ Остальные языки, в т. ч. низкоресурсные **— 3 %** (несколько сотен пар)



Что предлагается в работе?

→ Воспроизвести одно из представленных участниками решений полного дообучения модели mBERT (ресурсоемко, затратно по времени)

→ Дообучить модель mBERT с помощью адаптера LoRA (занимает считаные проценты параметров, обучается быстрее, возможна небольшая потеря точности)

→ Сравнить результаты

03 Что такое LoRA

LoRA не трогает основную матрицу весов: к ней добавляются две компактные матрицы A и B. Их произведение B*A образует низкоранговую матрицу ΔW, которая прибавляется к замороженной проекции W.

Именно A и B (доли процента от исходных весов) дообучаются, позволяя модели адаптироваться, не трогая основную матрицу.

Модель получает свободу подстроиться под новую задачу, не растеряв знаний, накопленных в предобучении.

Мой пайплайн: mBERT + полный fine-tune vs mBERT + LoRA

Сначала я воспроизвела стандартный скрипт Kaggle, полностью дообучив mBERT-base на задаче логического вывода. Затем на тех же данных запустила тот же mBERT, но с LoRA-вставками ранга 16 только в матрицах query и value

Механизм токенизации

Слово → Число (токен) → Эмбеддинг (числовой вектор)

Токены хранятся в векторной базе, чем ближе слова друг к другу, тем «похожее» их статистическое окружение в языке. Поэтому близость эмбеддингов отражает семантическую близость слов: cat и dog оказываются рядом, а cat и galaxy — далеко друг от друга.


Entailment
Neutral
Contradiction
Before LORA
After LORA

Гиперпараметры обеих моделей

Full fine-tune:

Эпохи = 3
LR = $2 \cdot 10^{-5}$,
batch = 16,
FP32

LoRA:

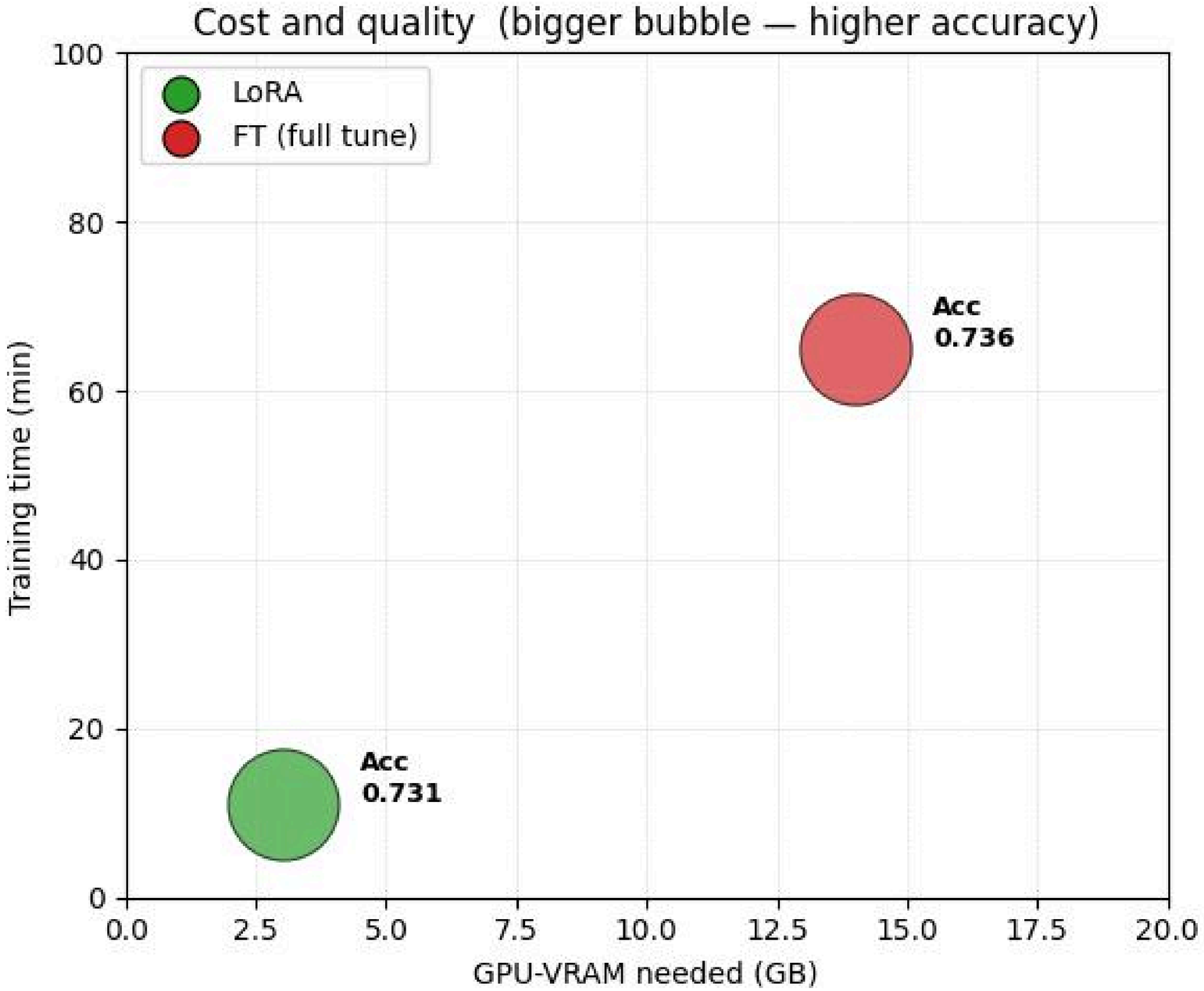
Эпохи = 3
LR = $5 \cdot 10^{-5}$,
физический batch = 8,
градиент
аккумулировался
два шага
FP16 (точность)

За один прямой-обратный проход через сеть на видеокарте обрабатывается 8 пар предложений

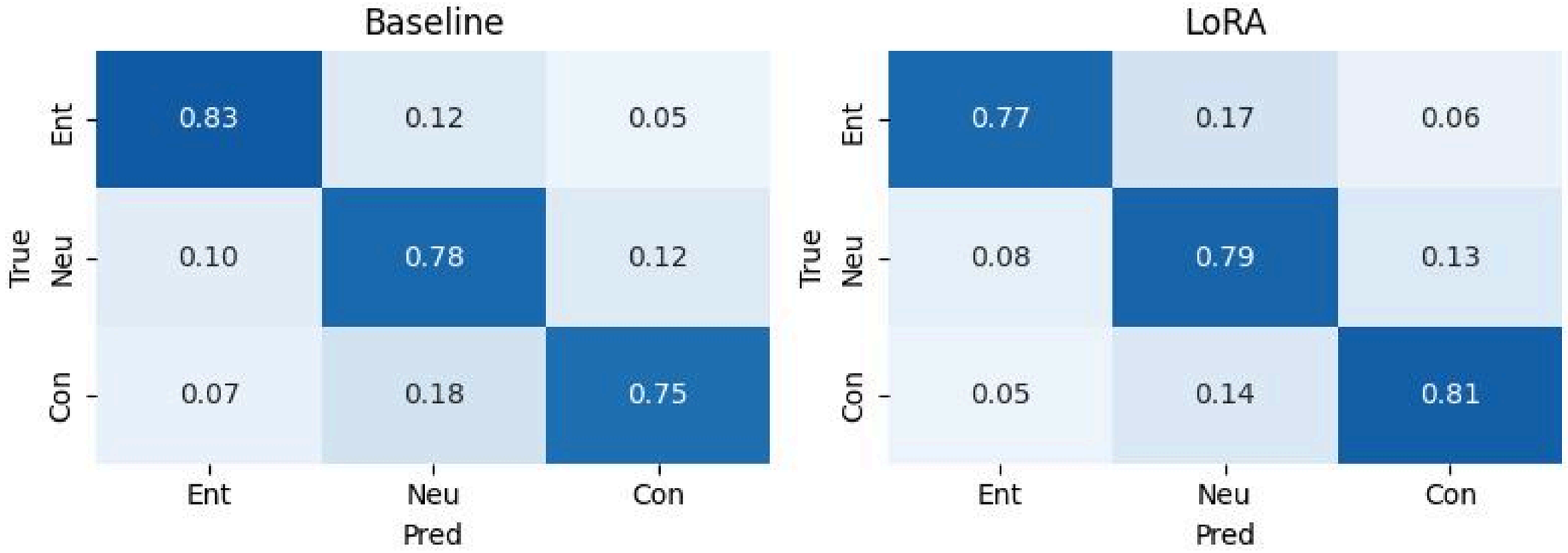
градиенты от двух таких batch по 8 складываются прежде чем обносить веса ⇒ эффективный размер батча получается 8 × 2 = 16

Метрики и ресурсы.

Полный fine-tune дал точность 0,736, потребляя 12,8 GB VRAM и 24 мин на эпоху;
LoRA показал точность 0,731 при 3,2 GB и 3.4 мин. Бар-диаграмма на соседнем слайде наглядно показывает экономию памяти и ускорение.

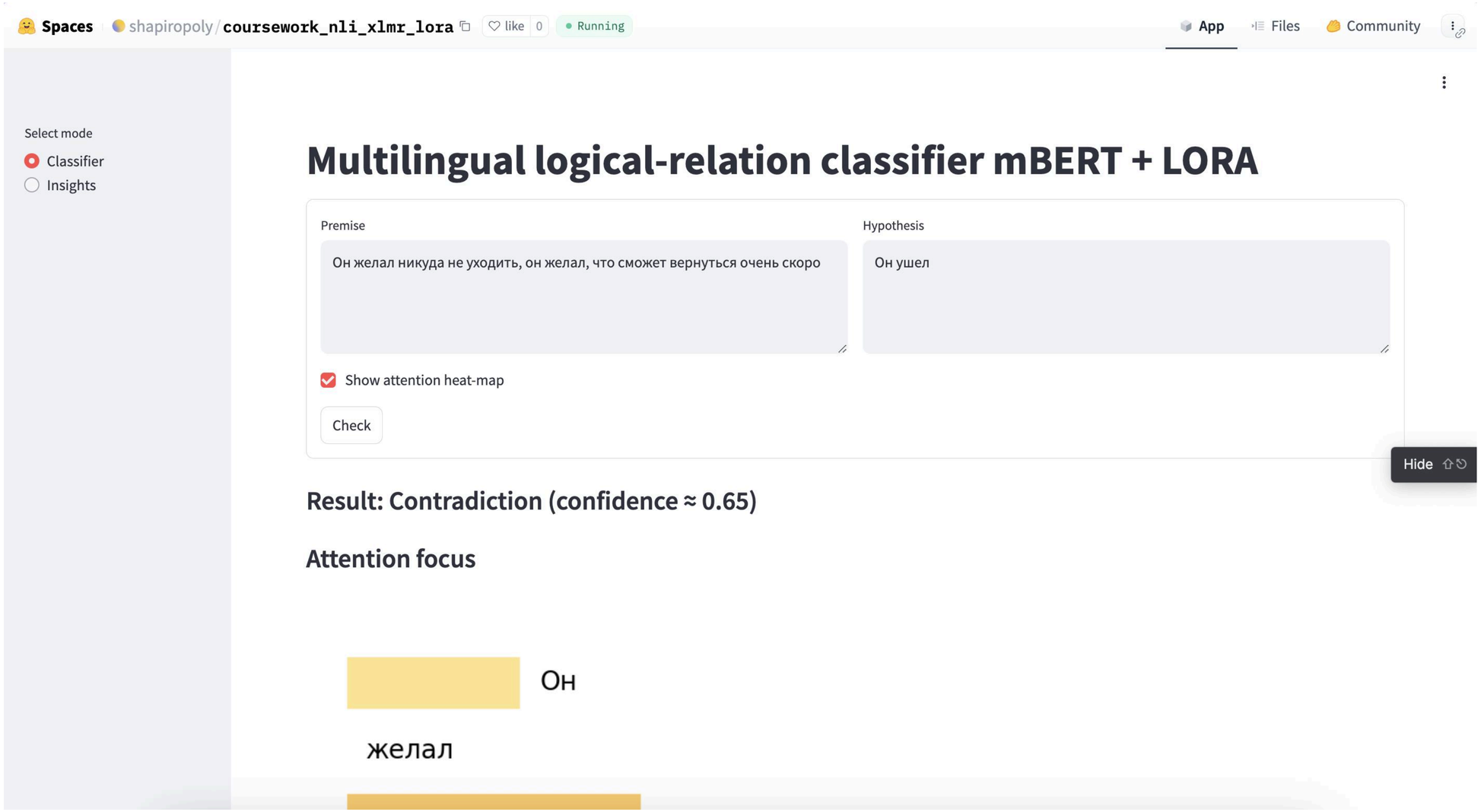

Cost and quality (bigger bubble — higher accuracy)
LoRA
FT (full tune)
Acc
0.736
Acc
0.731
Training time (min)
GPU-VRAM needed (GB)
0
20
40
60
80
100
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

Клетка (i → j) — доля примеров
истинного класса i (строка),
предсказанных как j (столбец).
Главная диагональ — recall класса.



Веб-интерфейс Streamlit + Hugging Face.

Модель загружена на Hugging Hub и обёрнут в одностраничное приложение: ввёл две фразы — получил метку и вероятность за 50 мс даже на CPU-сервере.

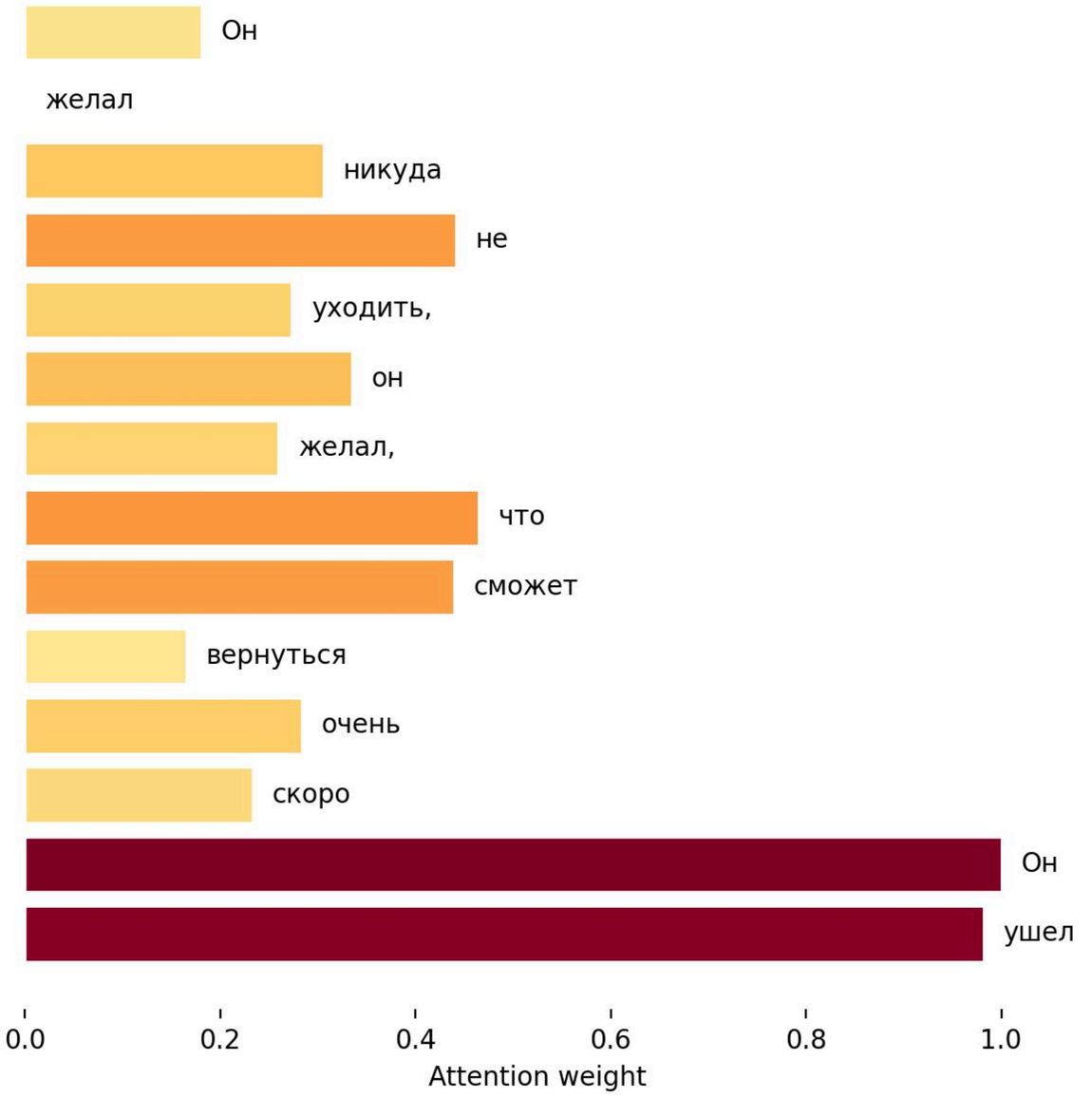


https://huggingface.co/spaces/shapiropoly/coursework_nli_xlmr_lora


Spaces
shapiropoly / coursework_nli_xlmr_lora
like 0
Running
Select mode
Classifier
Insights
Multilingual logical
LORA

Тепловая карта токенов в интерфейсе

При нажатии «Show attention» страница вычисляет, какие слова получили наибольшее внимание; чем ярче полоса, тем сильнее слово влияет на решение, благодаря чему пользователь видит, почему модель считает пары противоречивыми или согласованными.


Он
желал
никуда
не
уходить,
он
желал,
что
сможет
вернуться
очень
скоро
Он
ушел
0.0
0.2
0.4
0.6
0.8
1.0
Attention weight

Выводы

LoRA сохраняет 93–99 % точности полного fine-tune, но требует в четыре раза меньше видеопамяти и учится в 8 раз быстрее; а компактный чек-пойнт и веб-демо показывают, что многоязычный NLI-сервис можно поднять даже без GPU.

Стек

Библиотеки:

Tensorflow
Keras-NLP
Seaborn
Pandas

Среда обучения:

Google
Colab

Облако, контейнер, сервер:

Hugging
Face
Docker

Сет данных:

Kaggle

Язык:

Python

Веб-приложение:

Streamlit

Ссылка на веб-приложение

https://huggingface.co/spaces/shapiropoly/coursework_nli_xlmr_lora