Team 2

Turkish Joke Teller LLM Model

MALTEPE UNIVERSITY

SE 403 01 - Software Project Management

Team Members

Ali Eren Demireller

Ahmet Emir Solak

Berk Türk

Enver Yılmaz

Esra Aslı Aygın

Yavuz Behiç Aydoğmuş

Content

Team 2

Purpose

To produce a model good enough that it can tell original jokes in Turkish.

Planning

How did we organize ?

Organizing is challenging, especially in flat structures, needing everyone to take active role to help project progress towards success.

We encountered problems but in the end, we have managed to solve and make project come into reality

Planning & Communication

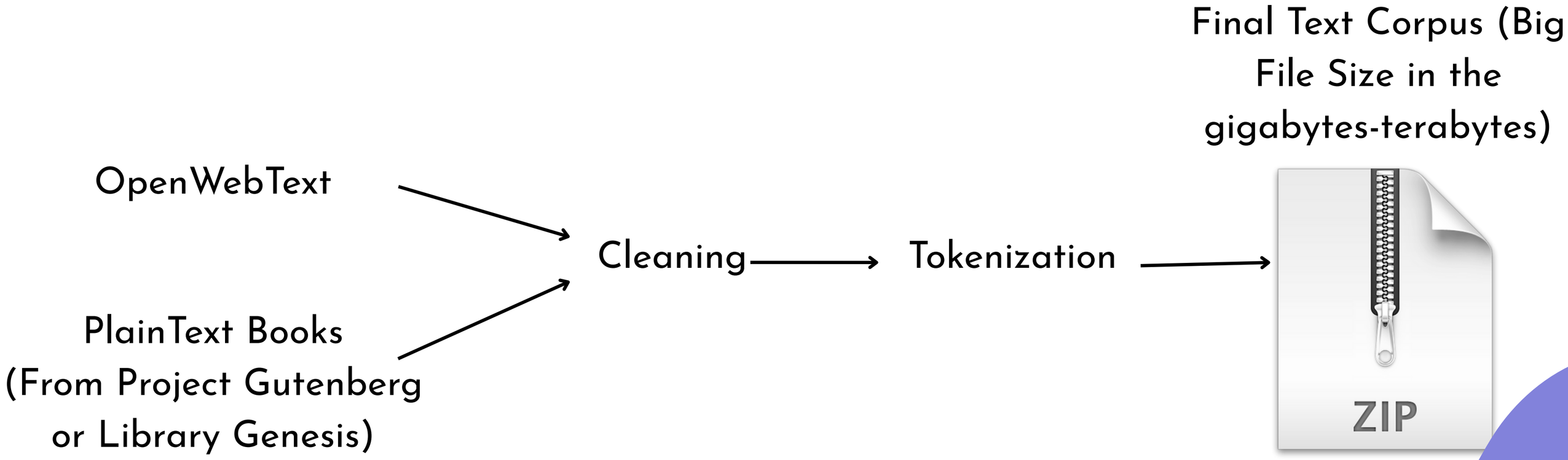
Agile workflow helped us to define goals in two weeks of cycles to focus better on smaller objectives.

Using Trello to define objectives, WhatsApp groups for short talks and Google Meet for longer talks helped us handle issues and get things done.

Having teacher acting as supervisor been helpful on setting new goals and helping us align on current work.

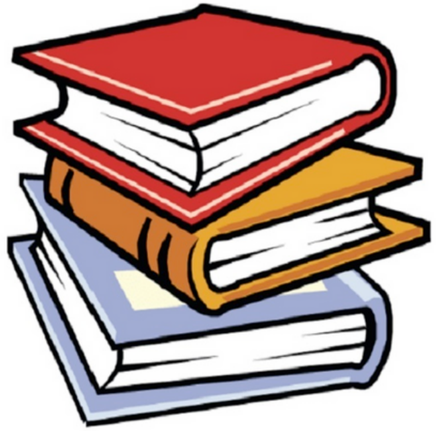
Dataset Collection

To train a large language model, you need to collect a huge amount of text. This isn't just random text. We need high-quality, diverse sources that reflect real human language. The more diverse the data, the better the model can understand and generate language.



Some of the most common data sources include websites from large web scrapes like OpenWebText, Wikipedia, plaintext books from Project Gutenberg, and sometimes even code from repositories like GitHub for coding models. But using data from the web brings up licensing and copyright issues so only certain content is legally usable.

OpenWebTextCorpus

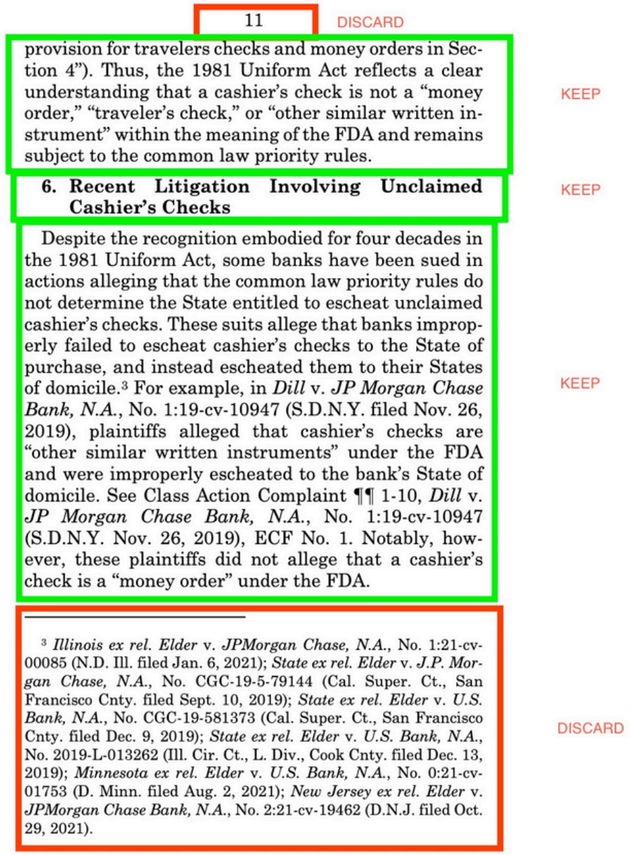


libgen.is

Project Gutenberg

WIKIPÉDIA
L'encyclopédie Libre

Once the data is collected, it's not ready yet. It’s often messy and unstructured. For example, converting a PDF book into plain text can be hard, headings might get mixed with paragraphs, or the formatting can break. Websites are full of ads, comments, or weird layouts that need to be cleaned up



You discard the page numbers and references when converting a pdf book into a plaintext file for a cleaner dataset.

I did this when extracting all “fikras” from fikra compilation books

After the data is cleaned, we tokenize it. Tokenization is the process of breaking text into smaller called tokens that the model can understand. These tokens are later converted into integers, which the model uses to learn patterns.

There are 3 types of tokenizers and I will briefly explain them:

1. Character-Level Tokenizer

Each character is converted into an integer. This gives us a tiny vocabulary and a very long sequence to process.

The model learns how letters are put together to form words.

It sees a lot of characters, which takes more time and memory.

Vocabulary size: very small (~100–500 characters)

2. Word-Level Tokenizer

Every word is a token. This leads to a huge vocabulary, possibly millions or billions if multilingual. “This is tokenizing.” becomes:

This - is - tokenizing - .

It has much fewer pieces, so it's faster.

But if you use a word that it hasn’t seen then you may get an error

3. Subword Tokenizer

Sits between character and word tokenizers. Words are broken down into smaller units (subwords or even characters) depending on how common certain patterns are in the training data. This method handles unknown words better and reduces vocabulary size while still capturing linguistic structure.

The word "unhappiness" might be split into subword tokens like:
"un", "happiness"
or
"un", "happi", "ness"

We used AutoTokenizer which is a subword tokenizer to make the dataset.

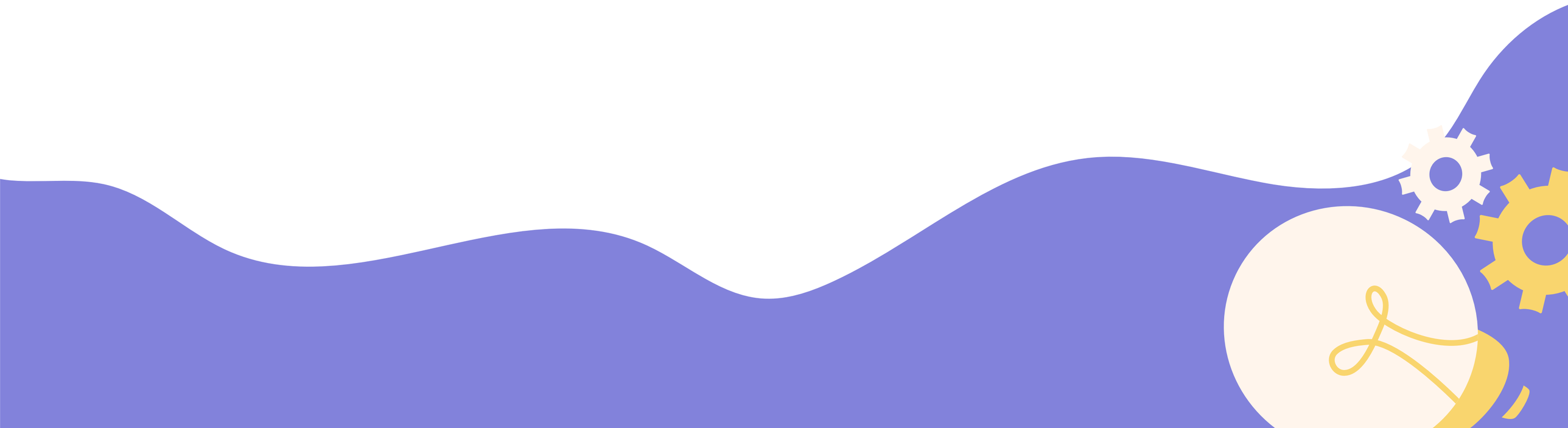
Pseudocode of how we did it:

```
import os
import json

input_folder = "books_plaintext"
output_json = "fikras_collection.json"

fikras = []

for filename in os.listdir(input_folder):
    if filename.endswith(".txt"):
        file_path = os.path.join(input_folder, filename)

        with open(file_path, "r", encoding="utf-8") as f:
            text = f.read().strip()

        fikras.append({
            "source_file": filename,
            "text": text
        })

with open(output_json, "w", encoding="utf-8") as f_json:
    json.dump(fikras, f_json, ensure_ascii=False, indent=2)

print(f"Saved {len(fikras)} fikras to '{output_json}'")
```

Pseudocode of how we did it:

```
import json
import numpy as np
from transformers import AutoTokenizer

input_json = "fikras_collection.json"
output_file = "tokenized_dataset.npy"

tokenizer = AutoTokenizer.from_pretrained("gpt2")

all_token_ids = []

with open(input_json, "r", encoding="utf-8") as f_json:
    fikras = json.load(f_json)

for fikra in fikras:
    text = fikra["text"]
    token_ids = tokenizer.encode(text, add_special_tokens=False)
    all_token_ids.extend(token_ids)

token_array = np.array(all_token_ids, dtype=np.uint16)
np.save(output_file, token_array)

print(f"Tokenized dataset saved to '{output_file}' with {len(all_token_ids)} tokens.")
```

Model Finetuning



Finetuning Stack & Data

Tech Stack

Google Collab via Jupyter Notebook
LoRA with Transformers Architecture on Torch

Data

Custom dataset resulted from teams
own data collection efforts.

Example Training Data For Finetuning

```
{
  "kategori": "Temel",
  "baslik": "Dursun pilav yapıyormuş. Temel sormuş",
  "metin": "Dursun pilav yapıyormuş. Temel sormuş: -Suyunu koydun mu? -Koydum. -Tuzunu koydun mu? -Onu da koydum. -Pilav nerede? -Onu koymayı unuttum!"
}
```

Finetuning Process

custom data

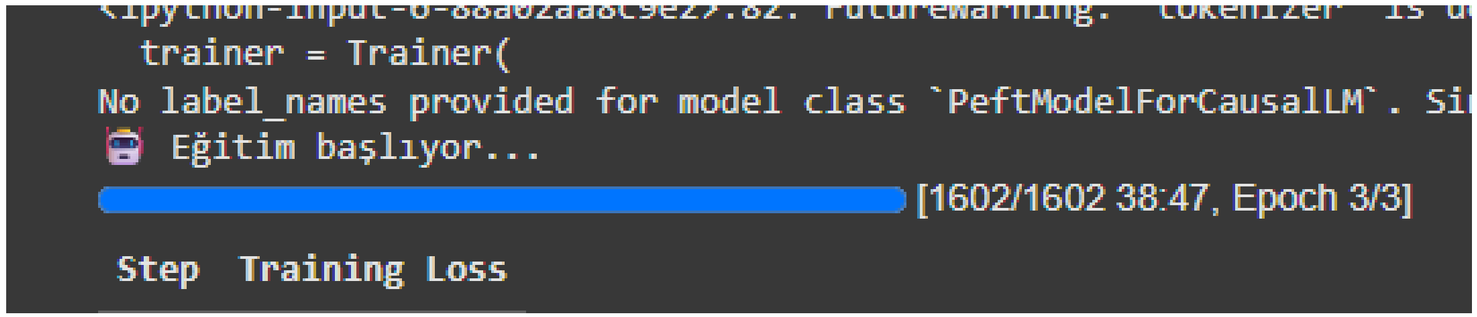
Base Model  Finetuned Model

LoRA

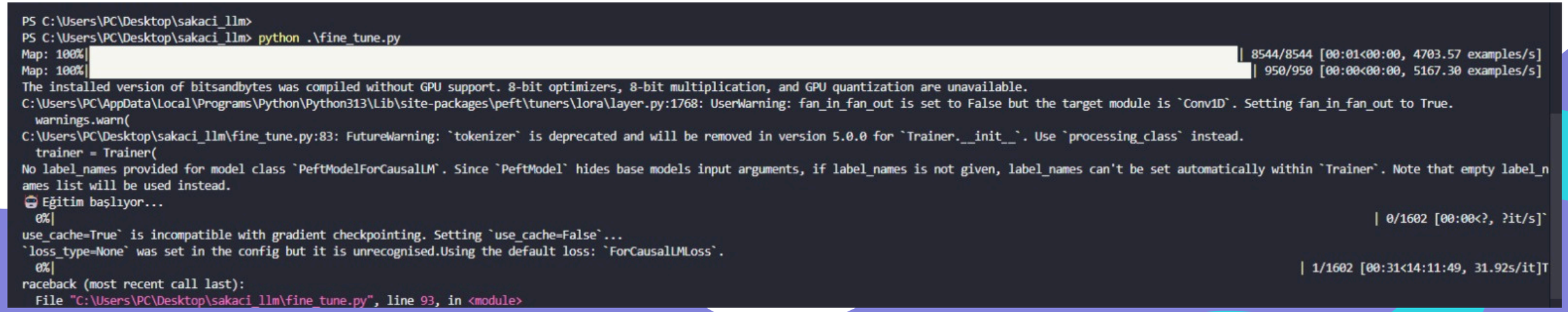
(Low Rank Adaptation)

Finetuning Process

Using 16GB T4 GPU Google Colab 35-40 min

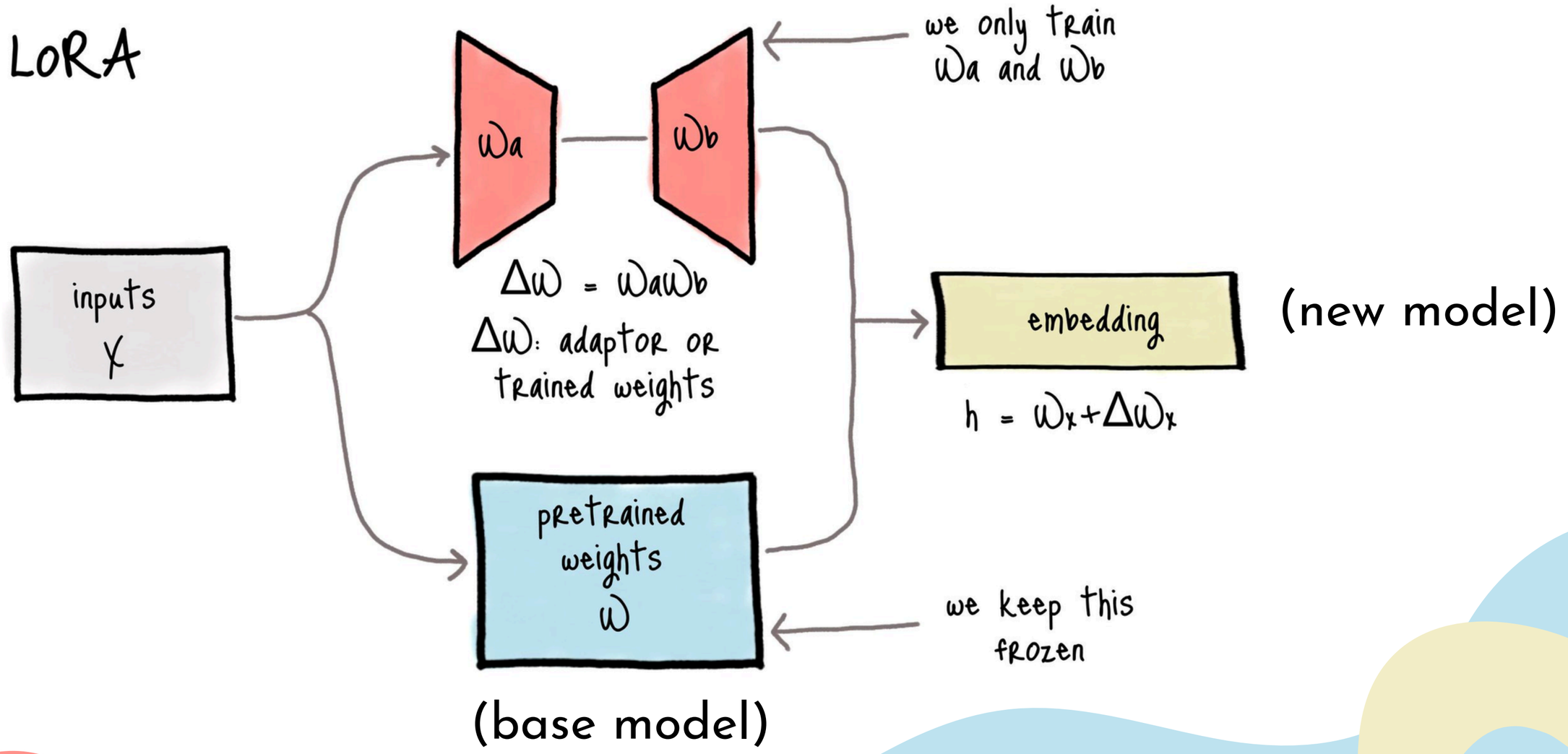


Using 6 GB RTX4050 GPU Laptop/PC 14-14.30h



Web Front-end

Finetuning with LoRA



Finetuned Model Example Result

Our Finetuned Models
Strengths & Weaknesses

Our polished model promotes clear and comprehensible Turkish usage, highlighting humor not through broad jokes, but through subtle, amusing remarks.

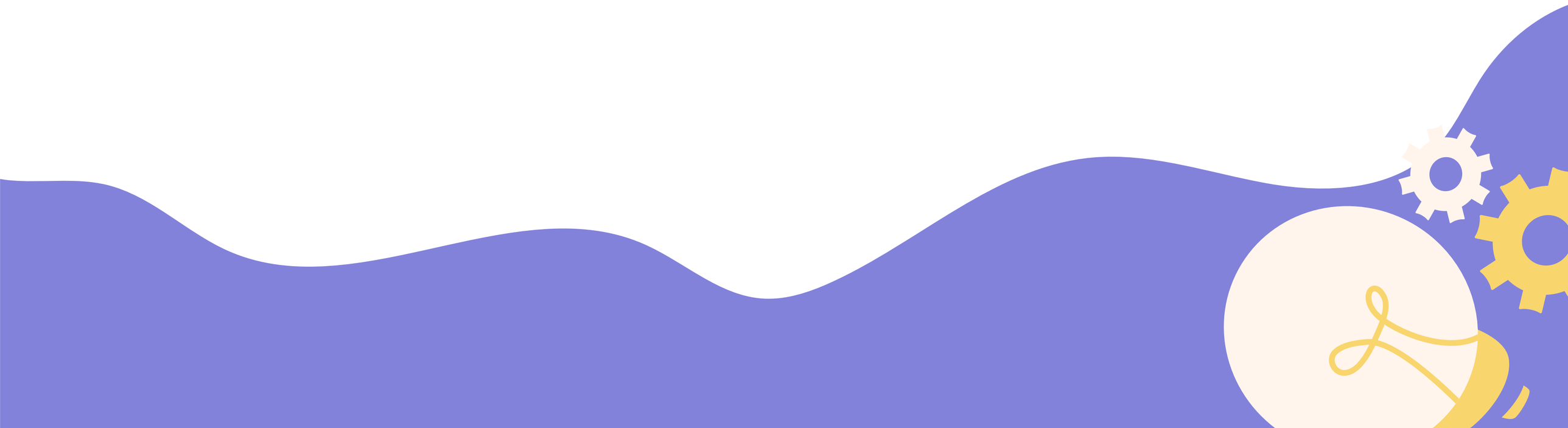
This model values the nuances of the Turkish language, ensuring that even the simplest phrases evoke a smile. Whether it's a lighthearted comment or a witty observation, it enriches interactions, making them more enjoyable and relatable. But weak on generating a coherent story.

Is Finetuning Worth It?

For short-form humour and chat-bot banter, the payoff easily justifies the finetuning effort.

Especially with it's adaptability, future work could target narrative coherence with a better base model and richer training data, but the current gains are already valuable.

Model from Scratch



Model Training

book_dataset.txt
wiki_turkce.txt
tr_others.txt

> 1.5 GB

Then

Fine-tuning

Example LSTM Fikra Data


```
{
 "title": "ADAM OLMAK",
 "content": "Bir gün Hoca nın bulunduğu bir sohbette sormuşlar: “Hocam, adam olmanın yolu nedir?” Hoca düşünceli düşünceli, başını bir o yana bir bu yana sallayarak “Söyleyen olursa dinlemeli, dinleyen olursa söylemeli” demiş."
},
```

Why LSTM?

LSTM is good at learning long-term dependencies in sequential data. This is ideal for tasks where context is important, such as text generation.

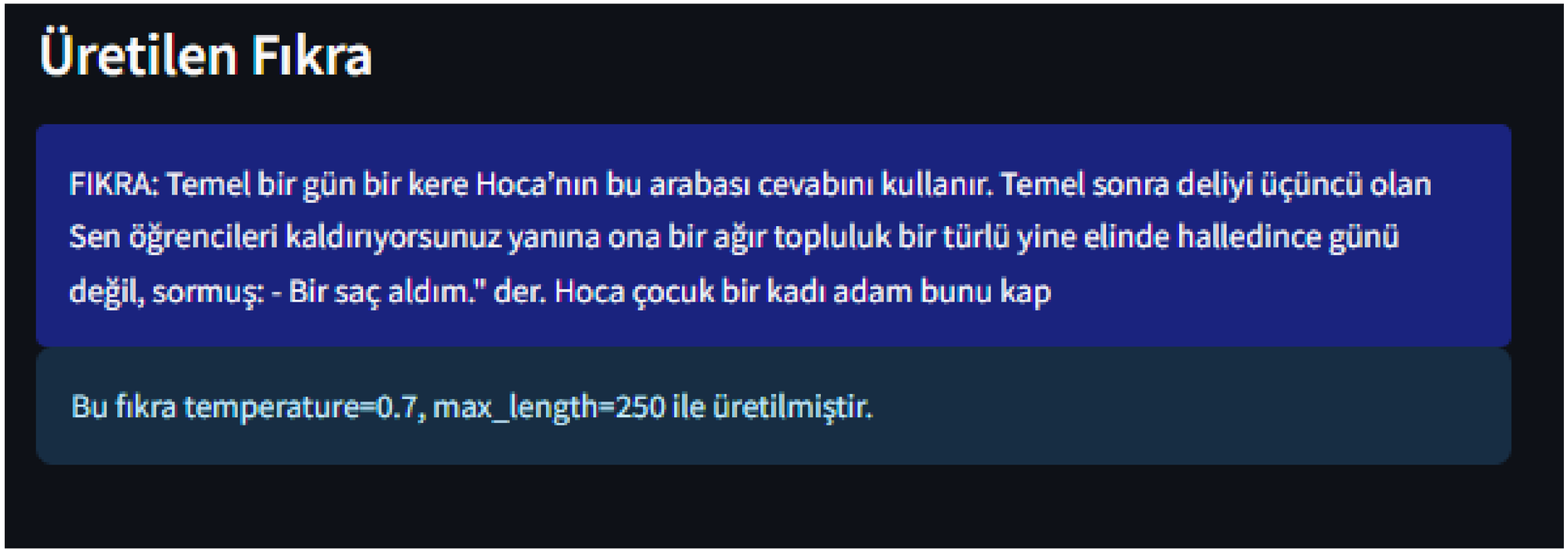
The reason LSTM is preferred over QA (Question-Answer) models is the need for a large amount of specially labeled data for QA. Since such a dataset is not available for joke generation, a generative model like LSTM is a more suitable choice.

Example QA Fikra Data

{
"question": "Bana temel ile ilgili bir fıkra anlatır mısın?",

"answer": "Temel bir gün doktora gitmiş. Doktor, 'Temel, sana bir iyi bir de kötü haberim var. Hangisini önce söyleyeyim?' demiş. Temel, 'Önce kötü haberi söyle' demiş. Doktor, 'Maalesef çok az ömrün kaldı.' demiş. Temel şaşırmış, 'Peki iyi haber ne?' diye sormuş. Doktor gülerek, 'Bugün çok güzel bir gün, tadını çıkar!' demiş."
}

Result:


Üretilen Fıkra
FIKRA: Temel bir gün bir kere Hoca'nın bu arabası cevabını kullanır. Temel sonra deliyi üçüncü olan Sen öğrencileri kaldırıyorsunuz yanına ona bir ağır topluluk bir türlü yine elinde halledince günü değil, sormuş: - Bir saç aldım." der. Hoca çocuk bir kadı adam bunu kap
Bu fıkra temperature=0.7, max_length=250 ile üretilmiştir.

Conclusion

Problems We Encountered

- Scarcity of open-source Turkish jokes datasets
- Limited access to large, high-quality corpora
- Hardware constraints for training and inference
- English-trained baseline made Turkish punch lines feel unnatural

How We Solved Each Problem

- **Scarcity of open-source Turkish data:** Scraped & cleaned Turkish jokes, from books and websites to build our own corpus.
- **Limited access to big, well-labeled datasets:** Merged many mini-datasets and used artificial data to bulk them up.
- **Hardware constraints:** Trained on Google Colab with quantization where possible.
- **English-trained baseline made Turkish punch lines feel off:** Switched to a better multilingual/Turkish base model for fine-tuning and fed it a larger Turkish-humor dataset; for from-scratch runs we added even more Turkish tokens.

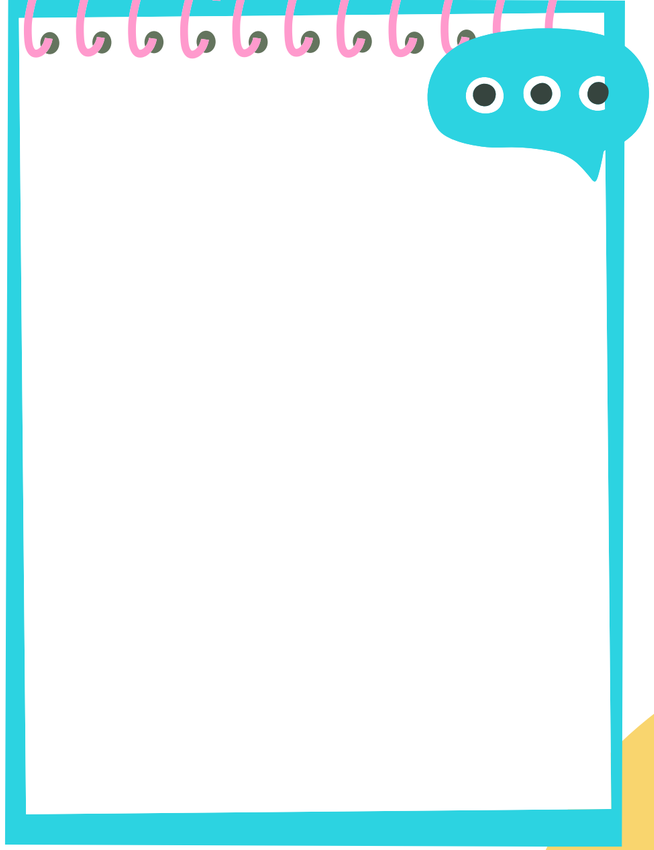
Future Potential

Potential of this project is massive due to it's future ability to cater for millions of Turkish speakers who would be greatly enjoy to hear and listen truly funny and original jokes that resembles their jokes and gives hints of their culture.

Even that ability can be replicated for other languages & cultures.

This models future version can be even used for:

- Producing social media content
- A system that produces jokes tailored to the user's interests or prior preferences.
- Creating engaging content for language learning or creative writing classes.



Key Takeaways

- Lesser quality on finetuning but rapid time-to-release.
- On finetuning, having to rely on the base model provider to innovate to have smarter model therefore lesser innovation and risk of increased dependence.
- Maintenance is painless in fine-tuning: swap-in new adapters as new model releases.
- Gathering specialized data is a time-consuming process, highlighting the importance of efficient data management strategies.

Final Thoughts

- Full stack control, that enables us to have grand control on which data we can feed
- Better possible safety checks through having a model from scratch
- Creating a model from scratch introduces stewardship burden – we now would have to deal with the full supply chain: data rights, GDPR, as well as to innovate ourselves.

Creating anything is pretty hard.

Team 2

Thank you for LISTENING

Maltepe University

SE 403 01 - Software Project Management