Turkish Joke-Telling AI Model Project Report

1. Scope of the Project

The primary objective of this project is to develop a Turkish joke-telling AI system by leveraging both **fine-tuning of a pre-trained large language model (LLM)** and experimenting with **training a custom joke generation model from scratch**. The project focuses on producing culturally relevant, humorous, and high-quality Turkish jokes using modern deep learning techniques.

The scope includes:

- **Fine-Tuning a Pre-Trained LLM**: The LLaMa 3.3 70B model will be fine-tuned on a curated dataset of Turkish "fıkra"s using the **LoRa method** implemented via the **Unsloth library**. This approach will adapt a powerful general-purpose model to the specific domain of Turkish humor.
- **Training a Model from Scratch (Optional/Exploratory)**: Alongside fine-tuning, the team will explore the feasibility of training a smaller LLM from scratch, exclusively on Turkish joke data. This experimental effort aims to assess whether a compact, domain-specific model can perform comparably in joke generation tasks.
- **Dataset Curation**: A unique dataset of Turkish jokes will be compiled and cleaned, focusing on cultural relevance, humor structure, and appropriateness.
- **Technology Stack**: The implementation will use **Python** and **PyTorch**, with training procedures orchestrated via the **Unsloth** ecosystem to ensure efficiency and modularity.

Out of Scope:

- **Multilingual Support**: The system will focus solely on Turkish jokes.
- **Audio/Voice Output**: Joke delivery will be in text form only.

This dual approach—fine-tuning a powerful existing model and experimenting with custom training—will provide insights into the adaptability of LLMs to niche domains such as culturally specific humor, and help define best practices for similar AI applications in other languages or creative domains.

2. Functionality of the Project

This project focuses on developing an AI system that generates culturally relevant Turkish jokes (“fıkra”) by leveraging large language model (LLM) technologies. The functionality evolved through two main development directions during the project lifecycle.

a. Functions Available at the Beginning of the Project

At the beginning of the project, the main functionality centered on:

- **Fine-tuning a pre-trained LLM**: The LLaMa 3.3 70B model was selected as the base model.
- **Domain adaptation with Turkish jokes**: A curated dataset of Turkish jokes was prepared to fine-tune the model using the LoRa method through the Unsloth library.
- **Core infrastructure setup**: Initial project setup involved Python and PyTorch environments, data cleaning scripts, and basic model training configurations.

This initial approach aimed to customize an existing model to the specific domain of Turkish humor.

b. Functions Added During the Development

As development progressed, the following new functionalities and directions were introduced:

- **Custom LLM Development from Scratch**: In addition to fine-tuning LLaMa, the team decided to experiment with building a smaller-scale language model from scratch trained entirely on Turkish joke data.
- **Dual Model Structure**: The system evolved into a hybrid structure where both the fine-tuned LLaMa model and the from-scratch model could be evaluated and compared.
- **Content Appropriateness Filtering**: Jokes were filtered to avoid offensive or culturally inappropriate content during both training and generation phases.
- **Prompt-Based Joke Generation**: A flexible generation method was implemented to allow joke generation based on user-provided themes or prompts.

3. Missing Parts

Despite significant progress, several functionalities or components initially planned or later considered during the development could not be implemented due to time and resource constraints:

• Web Interface (Full Deployment):

While the idea of deploying the joke-generating model via a web interface was considered, it was not finalized or deployed. The interface remained at a prototype or CLI level.

Reason: Time constraints and shifting focus toward model development and evaluation prevented front-end implementation.

• Extensive Dataset Expansion:

A broader and more diverse Turkish joke dataset was planned to improve model generalization. However, the dataset used remained relatively limited.

Reason: Difficulty in sourcing high-quality, diverse, and culturally appropriate Turkish jokes.

• Comprehensive Evaluation Metrics:

We thought to implement both automated and human evaluation frameworks to assess joke quality (e.g., funniness, cultural relevance, fluency).

Reason: Evaluation was performed informally and qualitatively, due to time limitations and lack of access to a large evaluator group.

• **API-based Delivery:**

Providing the joke generation functionality via a REST API was considered but not implemented.

Reason: Focus was kept on the model development and offline testing stages.

• **Full Training of a Large-Scale Custom LLM:**

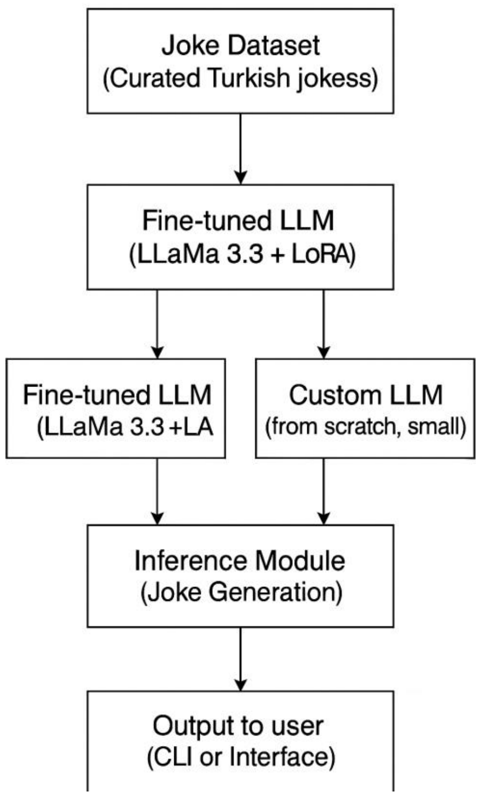
Although a basic from-scratch LLM was trained, it remained limited in scale and performance due to computational limitations.

Reason: Training a competitive large-scale LLM from scratch requires extensive computational resources beyond the project scope.

4. Design Documents

The project architecture was designed to support two main tracks of development: fine-tuning a pre-trained model (LLaMa 3.3 70B) and training a smaller custom LLM from scratch. Below is an overview of the key design components and system structure.

a) System Architecture Overview



b) Data Flow

1. **Input:** Dataset of Turkish jokes, preprocessed and filtered for quality and cultural appropriateness.
2. **Model Preparation:**
    - For fine-tuning: Pre-trained LLaMa 3.3 model adapted using LoRA via the Unsloth library.
    - For custom LLM: Initialized and trained from scratch using the same dataset.
3. **Training/Evaluation:** Both models were trained and evaluated offline.
4. **Joke Generation:** Input prompts used to generate jokes from the models; outputs were manually reviewed for humor and fluency.

c) Design Choices

- **Model Choice:** LLaMa 3.3 70B was chosen due to its performance in few-shot and instruction-based generation tasks.
- **LoRA Method:** Enabled efficient fine-tuning without the need to retrain the entire model, making it feasible within resource constraints.
- **Unsloth Library:** Provided memory-efficient implementations for handling large models and LoRA integration.
- **Python & PyTorch:** Used for model manipulation, dataset handling, and training loops.
- **Scalability Plan:** While initial deployment was limited, the architecture is modular enough to allow future integration with a front-end or REST API.

5. Deployment

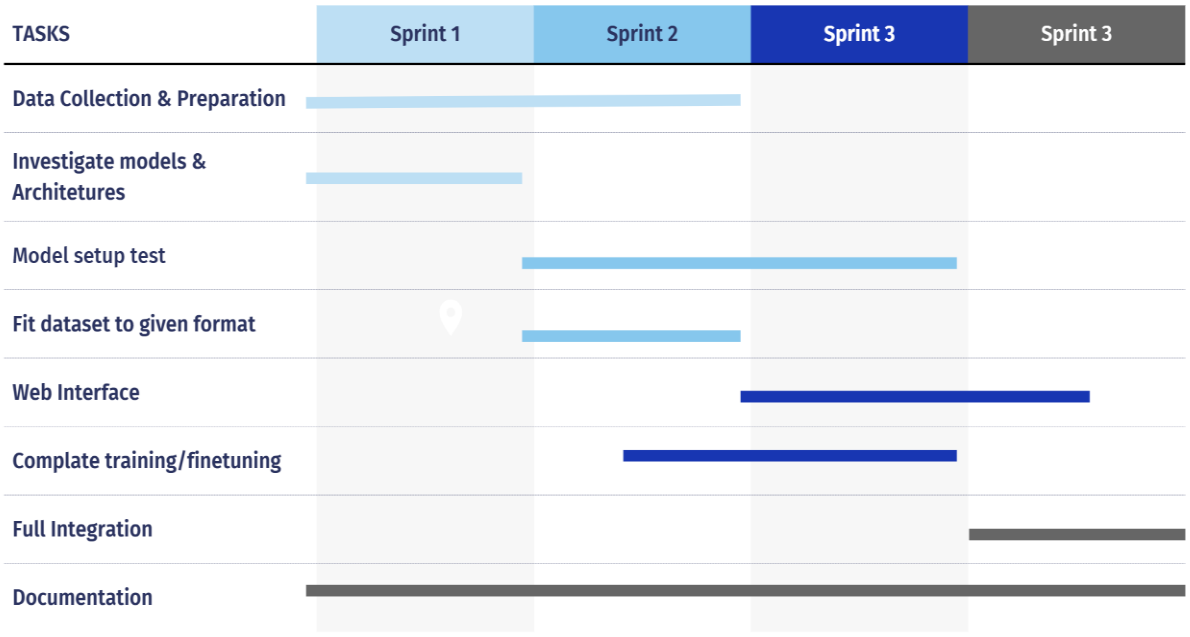
To deploy and run the software, follow these steps:

1. **Environment Setup**:
    - Install Python 3.10 or above.
    - Create and activate a virtual environment.
    - Install required libraries listed in `requirements.txt`.
2. **Model Training**:
    - For fine-tuning: Run the script `finetune_llama.py` with the Turkish joke dataset.
    - For training from scratch: Use `train_custom_llm.py` with proper configurations.
3. **Joke Generation**:
    - After training, use `generate_joke.py` to generate jokes from the trained model.
4. **Execution**:
    - All scripts are executed via command-line interface. No web interface is included.
5. **Hardware Note**:
    - Model training requires a GPU-enabled machine with high memory

6. Tasks and Responsibilities by Iteration

Trello link : https://trello.com/b/fdlne0hB/team-2

iter no/ developer	Ali Eren (leader)	Emir	Berk	Yavuz	Enver	Aslı
İter 1	Investigate models & architectures	Investigate models & architectures	Investigate models & architectures	Data Collection & Preparation	Data Collection & Preparation	Investigate models & architectures
İter 2	Start training/fine-tuning	Start training/fine-tuning	Start training/fine-tuning	Fit dataset to given format.	Fit dataset to given format.	
İter 3	Improve finetuning generation	Improve finetuning generation	Document preparation	Interface	Interface	
İter 4			Write LLM from scratch	Write LLM from scratch	Presentation preparation	



7. Risk Management

Several risks arose during the development of the project. One of the biggest concerns was the presence of culturally inappropriate or offensive jokes in the dataset. To address this, the data was carefully selected. Another major challenge was ensuring that the model understood the nuances of Turkish humor. We mitigated this by iteratively fine-tuning the model with updated datasets and closely evaluating the outputs.

Legal and ethical concerns regarding joke licensing were also taken into account. All joke content was either sourced from public domain materials or used with appropriate permissions. We also used data augmentation techniques and monitored joke uniqueness to avoid overfitting, where the model could repeat jokes from the training data.

The system was developed using a modular architecture to ensure future extensibility and avoid scalability issues.

8.Tests

The software was tested across three main dimensions: model output, user interface, and content relevance. For model behavior, we verified that the generated jokes were in Turkish, culturally appropriate, and not memorized from training data. We also checked that the model's response time stayed within acceptable limits.

For the user interface, we ensured that jokes were properly displayed upon user interaction, providing immediate and clear feedback to users. In terms of content quality, we evaluated whether the jokes reflected culturally relevant themes—such as Turkish names, settings, and traditions.

Testing was performed manually by interacting with the interface and analyzing the outputs, given the experimental nature of the project. While no automated testing frameworks were used at this stage, all cases were defined clearly and reviewed during each update to validate functionality and quality.

9. Experience gained

Ahmet Emir Solak: During this project, I was responsible for the fine-tuning stage of a large language model. I learned how to adapt a pre-trained model to a specific dataset using low-resource optimization methods like LoRA. This process taught me how powerful language models can be tailored to perform better in specialized tasks, such as generating contextually appropriate and creative jokes. I also gained hands-on experience with tools like Hugging Face Transformers and Colab, and learned how to handle real-world challenges like hardware limitations and dataset formatting. Overall, it helped me better understand how language models can be customized and applied effectively.

Enver Yılmaz: I took part in the data collection and analysis processes of the project. I had the opportunity to work on data formatting. We also worked on a basic user interface for the trained LLM model. Throughout this process, I gained experience in both research and interface design. While I usually take on a team leader role in my projects, in this one I stepped back from management and focused on completing the tasks assigned to me, which provided me with a different kind of experience.

Berk Türk: In this project, I was responsible for researching suitable language model architectures and assisted in selecting the most suitable model for our task. I researched state-of-the-art models, focusing on those that support efficient fine-tuning, such as LLaMA 3. In parallel, I prepared documentation to clearly explain our methodology and technical setup. As the project progressed, I started building a language model from scratch to gain a deeper understanding of tokenization, model structure, and training dynamics. This gave me hands-on experience with the inner workings of LLMs and strengthened my skills in model architecture, optimization, and practical deployment.

10. Source code repository

https://github.com/Maltepe-University-SWEng/term-project-team-2