JOKE BOT MODEL PROJECT REPORT

Project Details	
University	Maltepe University
Course	SE 403 01 Software Project Management
Teacher	Ensar Gül

Project Participants	
Full Name	**Student Number**
Alp Salcıoğlu	220706016
Esma Şen	210706041
Berke Ensel	210706039
Melahat Eda Acar	210706038
İbrahim Kartal	210706004
Eren Yıldırım	230706301

TABLE OF CONTENTS

1. Scope of the Project

This project aims to develop a joke generation platform leveraging large language models (LLMs). It includes two distinct sub-projects: one using a custom-trained Transformer model trained on a diverse joke dataset, and the other using a fine-tuned version of the Turkish GPT-2 model specifically adapted to generate Nasreddin Hoca jokes. Both models are deployed using user-friendly interfaces (one with Node.js & EJS, the other with Streamlit) to ensure accessibility and real-time generation.

The main goal is to assess the effectiveness of both approaches in generating contextually appropriate, coherent, and culturally meaningful jokes in Turkish. The results are integrated into a user-facing web interface via Streamlit, allowing real-time generation of jokes through a simple interactive UI.

A key theoretical foundation of this project is the Transformer architecture, which forms the backbone of modern LLMs. Our understanding and application of this architecture were significantly influenced by the seminal paper *“Attention is All You Need”* (Vaswani et al., 2017). This work introduced the self-attention mechanism and a fully attention-based model architecture, which we adopted in the design and training of our custom joke-generation model. The insights gained from the paper were instrumental in shaping the model’s structure, training strategy, and performance expectations.

From a software engineering standpoint, the project adheres to modern software project management principles, including clear scope definition, version control (GitHub), risk analysis, documentation, frontend-backend integration, and iterative development.

1.1 Primary Objective

Custom Model Objective:

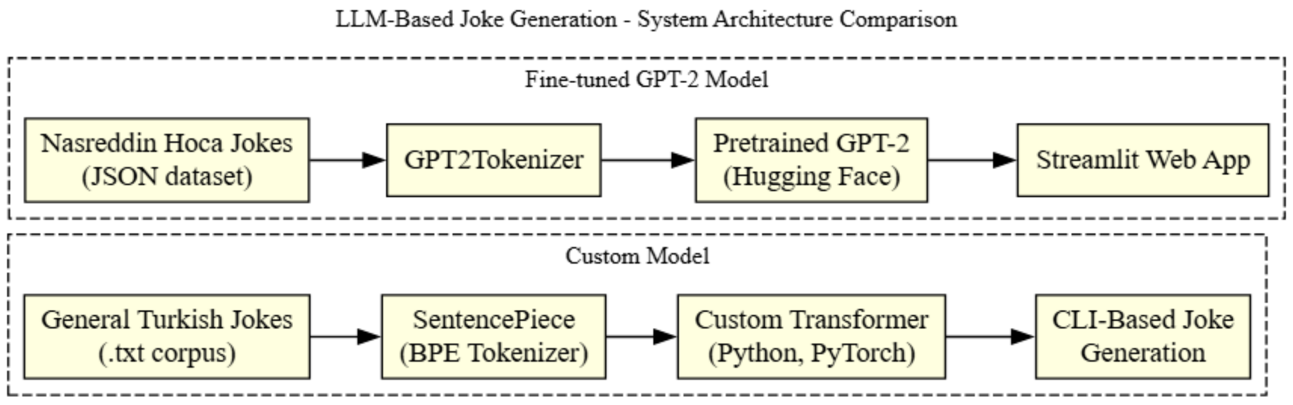
The main goal of developing the custom Transformer-based model was to build a foundational language model from scratch, capable of generating general Turkish jokes. Unlike pre-trained models, this model was trained on a limited yet diverse collection of Turkish jokes using a custom tokenizer built with SentencePiece. Through this effort, the project team aimed to gain a deep understanding of the fundamental elements of LLM development—such as vocabulary design, model architecture, and training loop implementation.

The objective was not only to generate jokes but also to create a fully self-managed pipeline that reflects the constraints and decisions involved in training from scratch. Additionally, it served as an experimental benchmark to compare with more advanced techniques like fine-tuning. This model emphasized learning-by-building and offered insights into token-level predictions, loss dynamics, and generation patterns of small-scale LLMs.

Fine-tuned Model Objective:

The fine-tuned model focused on enhancing the capabilities of a pre-trained Turkish GPT-2 model by adapting it to a specific cultural niche: Nasreddin Hoca jokes. The primary aim was to transfer the general language knowledge encoded in the GPT-2 weights and specialize it using a domain-specific dataset.

This allowed the model to better mimic the unique style, tone, and humor characteristic of Nasreddin Hoca stories. The fine-tuning process prioritized stylistic alignment, increased coherence in storytelling, and culturally relevant humor generation. By integrating the fine-tuned model with an interactive Streamlit interface, the project provided users with a real-time, one-click Nasreddin Hoca joke generator—combining LLM intelligence with user-friendly deployment.


LLM-Based Joke Generation - System Architecture Comparison
Fine-tuned GPT-2 Model
Nasreddin Hoca Jokes (JSON dataset)
GPT2Tokenizer
Pretrained GPT-2 (Hugging Face)
Streamlit Web App
Custom Model
General Turkish Jokes (.txt corpus)
SentencePiece (BPE Tokenizer)
Custom Transformer (Python, PyTorch)
CLI-Based Joke Generation

2. Functionality of the Project

This project integrates two different AI models for Turkish joke generation: a custom-built Transformer model and a fine-tuned GPT-2 model. Together, they form a complete text generation system capable of producing culturally resonant jokes in Turkish. Each model was developed with distinct architectural and functional goals, which contributed to the richness of the final product.

2.1 Overview of the Software's Primary Features and Capabilities

Custom Model (General Turkish Jokes)

The custom model was built from scratch using PyTorch and was trained on a dataset of general Turkish jokes. It features a SentencePiece-based tokenizer and a manually implemented Transformer architecture. The model generates short, grammatically valid jokes through a terminal-based interface.

Key Capabilities:

- Custom Transformer architecture with positional encoding, multi-head self-attention, and embedding layers.
- Custom tokenizer trained using the BPE (Byte Pair Encoding) method via SentencePiece.
- Lightweight model suitable for training in limited-resource environments.
- Command-line based joke generation for testing and experimentation.

Unique Attributes:

- Fully self-developed without any pretrained components.
- Educational and experimental value for understanding LLM internals.

Fine-Tuned Model (Nasreddin Hoca Jokes)

The fine-tuned model is based on the ytu-ce-cosmos/turkish-gpt2 pre-trained GPT-2 model. It was fine-tuned on a curated dataset of Nasreddin Hoca jokes and integrated into a web-based application using Streamlit.

Key Capabilities:

- Natural and stylistically coherent Nasreddin Hoca fıkra generation.
- Hugging Face Transformers pipeline for simple and flexible inference.
- Interactive Streamlit interface with a "Generate Joke" button.
- Tunable generation parameters (max_length, top_k, top_p, temperature) for customization.

Unique Attributes:

- Seamless deployment through a user-friendly interface.
- Domain-specific fluency and cultural humor precision.
- Demonstrates efficiency of transfer learning over training from scratch.

2.2 Functions Available at the Beginning of the Project

Custom Model:

- Raw .txt dataset of general Turkish jokes.
- Tokenization pipeline using SentencePiece.
- Early Transformer structure (single-layer prototype).
- Initial training loop with static learning rate.
- Manual inference via Jupyter Notebook (token-to-text decoding).

Fine-Tuned Model:

- Access to pre-trained Turkish GPT-2 (ytu-ce-cosmos/turkish-gpt2) via Hugging Face.
- Text-generation using Hugging Face's pipeline() function.
- Notebook-based manual input-output generation.
- Console output only, no graphical UI.

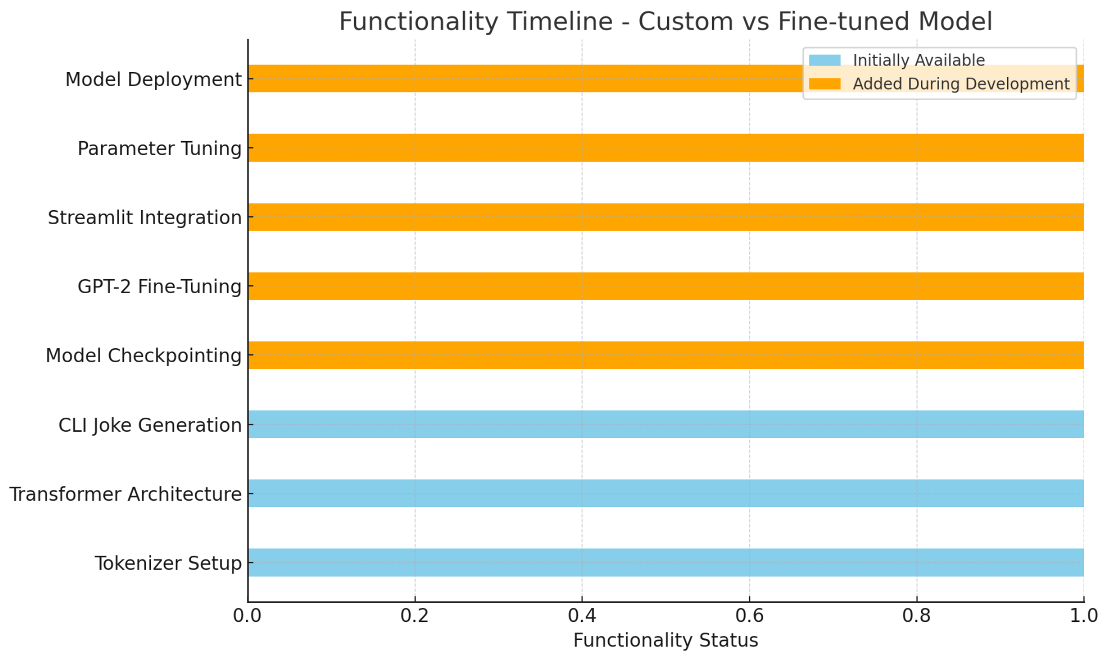
2.3 Functions Added During the Development

Custom Model:

- Finalized multi-layer Transformer with positional embeddings.
- Tokenizer serialization and reuse via .model file.
- Implementation of training metrics logging (loss, accuracy).
- Model checkpoint saving (.pth) for reusable inference.
- Comparison scripts to evaluate model behavior with various prompt lengths.

Fine-Tuned Model:

- Dataset cleaning and formatting into JSON for supervised learning.
- Fine-tuning on cleaned Nasreddin Hoca joke dataset.
- Integration with Streamlit web interface for real-time interaction.
- Hyperparameter tuning for more engaging and fluent outputs
- External osting of .bin model file and integration instructions in README.Evaluation scripts to compare outputs with the custom model.



3. Missing Parts

Despite the project's successful development and deployment of both a custom Transformer model and a fine-tuned GPT-2 model, there were several functionalities that were originally planned but not fully realized due to time constraints, resource limitations, or scope adjustments during the development process.

3.1 Unimplemented Features

1. **Integrated Multi-Model Interface**
   A single unified web interface where users could toggle between the custom model and the fine-tuned GPT-2 model to compare results in real-time.
2. **Evaluation Module**
   A joke evaluation mechanism using either rule-based scoring (e.g., language fluency, sentence length, token repetition) or external LLM-based scoring.

3. **User Feedback Collection System**
   A feedback form or rating system where users could rate generated jokes. These ratings would later be used for reinforcement fine-tuning or dataset refinement.
4. **Automated Testing Framework**
   A test suite that could verify the quality of outputs under different prompt conditions (e.g., minimal input, long prompt, empty input, etc.).

3.2 Reasons for Exclusion

- **Time Constraints:**
  Given the academic calendar and workload per iteration, several advanced features—such as model switching and evaluation modules—had to be deprioritized.
- **Resource Limitations:**
  The custom model required minimal resources, but the fine-tuned model's .bin file exceeded GitHub's upload limit and required external hosting. Cloud deployment was avoided due to GPU requirements.
- **Scope Management:**
  The primary goal was to build and compare two LLM-based models for Turkish joke generation. Therefore, UI enhancements and analytics modules were considered out of scope for this version.
- **Team Size:**
  With only three contributors, parallel work was required. Priority was given to core model development, dataset preparation, training, and deployment, leaving advanced features for a potential future version.

4. Design Documents

This section outlines the architectural and design aspects of the Joke Generation System, developed in two primary tracks: a custom-built Transformer model and a fine-tuned GPT-2 model. Each track had its own model logic, tokenizer pipeline, and frontend/backend integration strategies.

4.1 Model Architecture Design

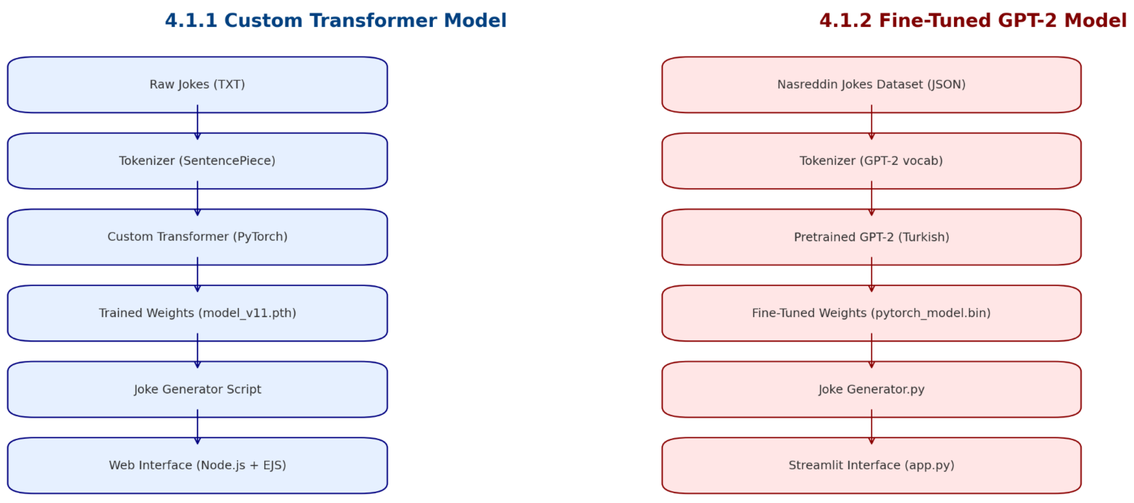
4.1.1 Custom Transformer Model

- **Architecture:** A Transformer-based language model developed from scratch in PyTorch.
- **Layers:** Multi-head attention layers followed by position-wise feedforward networks.

- **Tokenization:** Implemented with SentencePiece using a BPE vocabulary trained on general Turkish joke corpus.
- **Training Data:** General Turkish jokes (non-Nasreddin-specific), tokenized and encoded into custom .pth and .model files.
- **Model File:** model_v11.pth
- **Training Pipeline:** Managed within a Jupyter notebook (model.ipynb) including training loop, optimizer, loss calculation, and checkpointing.

4.1.2 Fine-Tuned GPT-2 Model

- **Base Model:** Pretrained Turkish GPT-2 (ytu-ce-cosmos/turkish-gpt2).
- **Fine-Tuning Dataset:** Nasreddin Hoca jokes cleaned and structured into a JSON dataset (nasreddin_hoca_jokes.json).
- **Training Output Files:**
    - pytorch_model.bin
    - tokenizer_config.json
    - training_args.bin
    - vocab.json, merges.txt
- **Training Script:** Fine-tuning performed in nasreddinn.ipynb.
- **Tokenizer:** Hugging Face AutoTokenizer, further refined for Nasreddin style.


4.1.1 Custom Transformer Model
Raw Jokes (TXT)
Tokenizer (SentencePiece)
Custom Transformer (PyTorch)
Trained Weights (model_v11.pth)
Joke Generator Script
Web Interface (Node.js + EJS)
4.1.2 Fine-Tuned GPT-2 Model
Nasreddin Jokes Dataset (JSON)
Tokenizer (GPT-2 vocab)
Pretrained GPT-2 (Turkish)
Fine-Tuned Weights (pytorch_model.bin)
Joke Generator.py
Streamlit Interface (app.py)

4.2 Tokenizer Configuration

This section details how tokenization was handled in both the custom-built Transformer model and the fine-tuned GPT-2 model. Tokenizers are essential for converting input text into numerical token IDs and significantly impact a model's performance, especially for morphologically rich languages like Turkish.

4.2.1 Custom Model Tokenizer (SentencePiece)

The custom model utilizes a tokenizer trained using the SentencePiece library. This approach was selected due to its ability to learn unsupervised subword tokenization without requiring pre-tokenized data. It is particularly effective for low-resource and domain-specific datasets.

Details:

- Tokenizer Type: Byte Pair Encoding (BPE)
- Training Tool: spm_train (SentencePiece CLI)
- Corpus: General Turkish joke dataset (.txt file)
- Vocabulary Size: 5000 tokens
- Model File: fikra_tokenizer.model
- Vocabulary File: fikra_tokenizer.vocab
- Training command used:

```
spm_train --input=cleaned_jokes.txt --model_prefix=fikra_tokenizer --vocab_size=5000
--model_type=bpe
```

Integration:

- The trained tokenizer model is loaded using SentencePiece's Python wrapper.
- Tokenized outputs are passed directly to the PyTorch Transformer model during training and inference.
- The tokenizer is saved and reused during inference to ensure consistent token mapping.

Advantages:

- Domain-specific vocabulary
- Effective subword splitting for Turkish
- Lightweight and fast

4.2.2 Fine-Tuned GPT-2 Tokenizer (Hugging Face)

The fine-tuned model uses the default tokenizer from the pretrained GPT-2 model ytu-ce-cosmos/turkish-gpt2. Hugging Face's tokenizer framework handles subword tokenization using a pre-trained vocabulary.

Details:

- Tokenizer Type: GPT-2 Byte-Pair Encoding (BPE)
- Library: Hugging Face Transformers
- Tokenizer Loading:

```
from transformers import GPT2Tokenizer
tokenizer = GPT2Tokenizer.from_pretrained("ytu-ce-cosmos/turkish-gpt2")
```

Files Generated:

- tokenizer_config.json
- special_tokens_map.json
- vocab.json
- merges.txt

Integration:

- Directly used in the Hugging Face Trainer class and in the pipeline() function for generation.
- Maintains compatibility with pretrained weights.

Advantages:

- Rich vocabulary trained on a large Turkish corpus
- Seamless integration with the GPT-2 architecture
- Automatic padding, truncation, and special token handling

4.3 Frontend and Streamlit Layout

The project includes two distinct user interfaces—one for the **custom Transformer model** built with a Node.js + EJS frontend, and another for the **fine-tuned GPT-2 model** using a Streamlit web application. Both interfaces were designed to facilitate easy interaction with their respective joke-generation backends.

4.3.1 Node.js + EJS Interface (for Custom Model)

This frontend was developed to allow user interaction with the custom-trained model hosted on a Python backend. The system includes authentication features, routing, form handling, and AJAX-based interactions using Axios.

Structure Overview:

- **Framework**: Node.js with Express and EJS (Embedded JavaScript templates)
- **Routing**: /register, /login, /profile, /jokes, /chat
- **Session Management**: Handled using express-session and jsonwebtoken
- **Form Validation**: Server-side using custom middleware
- **Database**: MongoDB via Mongoose (for user credentials)

Core Views (in views/ folder):

- register.ejs: User registration form
- login.ejs: Login page with email/password fields
- profile.ejs: View and update user information
- chat.ejs: Main joke generator page with prompt input
- jokeDisplay.ejs: Dynamically rendered joke output

Use Case Diagram

The use case diagram models the functional interactions between the user and the system. In the current implementation, the system supports only two operations:

- **Submit Prompt**: The user provides a sentence or topic as input to the language model.
- **View Generated Joke**: The model returns a joke, which is displayed to the user.

There is no user authentication, profile, or rating functionality in this version.

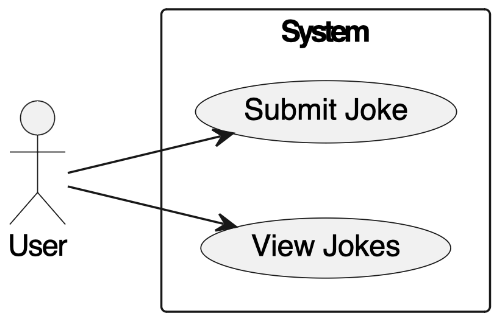


Figure 1 – Use Case Diagram

The design phase of the project played a crucial role in ensuring a structured, scalable, and maintainable architecture. In this project, which focuses on generating Turkish jokes using a language model, two core design diagrams were created: the Entity Relationship Diagram (ERD) and the Use Case Diagram.

Entity Relationship (ER) Diagram

Since the system is primarily built around taking a prompt and generating a joke, the database structure is minimal. The only active entity in the current version is:

- **Joke**: Represents the joke content generated by the model. Each joke has a unique id, a prompt field representing user input, and a generated_text field containing the model's output.

Although user interaction exists through prompt submission, the system does not include a persistent User entity in its current implementation.

See the figure below for the simplified ERD:

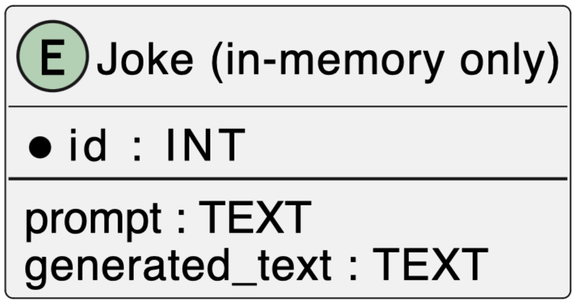
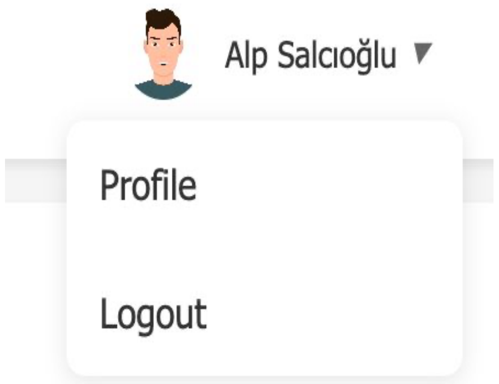
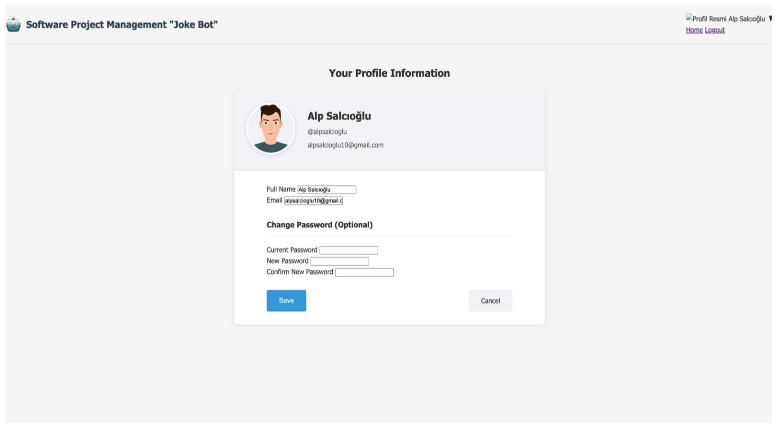


Figure 2 - ER Diagram

Note: The "Joke" entity is stored in-memory and not persisted in a database. This decision was made to reduce system complexity and focus on model interaction rather than long-term data management.




Software Project Management "Joke Bot"

Profil Resmi Alp Salcıoğlu ▼
Home Logout

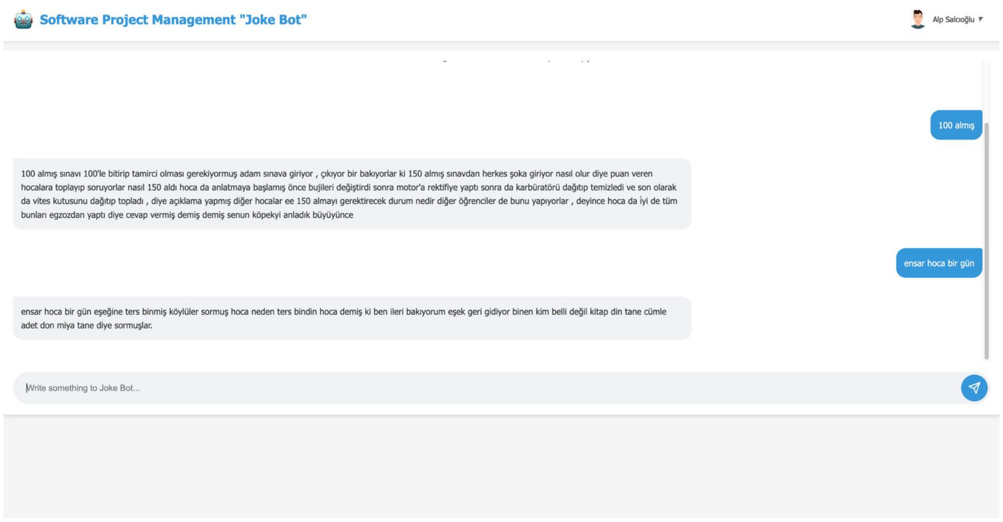
Your Profile Information

Alp Salcıoğlu
@alpsalcioglu
alpsalcioglu10@gmail.com

Full Name: Alp Salcıoğlu
Email: alpsalcioglu10@gmail.c

Change Password (Optional)

Current Password
New Password
Confirm New Password


Software Project Management "Joke Bot"

Alp Salcıoğlu

100 almış

100 almış sınavı 100'le bitirip tamirci olması gerekiyormuş adam sınava giriyor , çıkıyor bir bakıyorlar ki 150 almış sınavdan herkes şoka giriyor nasıl olur diye puan veren hocalara toplayıp soruyorlar nasıl 150 aldı hoca da anlatmaya başlamış önce bujileri değiştirdi sonra motor'a rektifiye yaptı sonra da karbüratörü dağıtıp temizledi ve son olarak da vites kutusunu dağıtıp topladı , diye açıklama yapmış diğer hocalar ee 150 almayı gerektirecek durum nedir diğer öğrenciler de bunu yapıyorlar , deyince hoca da İyi de tüm bunları egzozdan yaptı diye cevap vermiş demiş demiş senun köpekyi anladık büyüyünce

ensar hoca bir gün

ensar hoca bir gün eşeğine ters binmiş köylüler sormuş hoca neden ters bindin hoca demiş ki ben ileri bakıyorum eşek geri gidiyor binen kim belli değil kitap din tane cümle adet don miya tane diye sormuşlar.

Write something to Joke Bot...

Figure 3 - Custom Model Frontend

4.3.2 Streamlit Frontend Layout

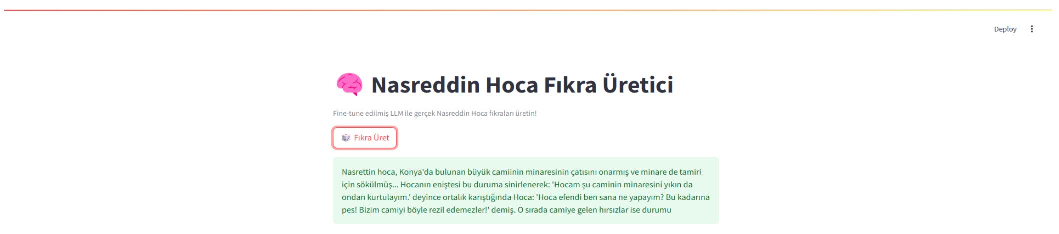
The Streamlit-based frontend was designed to offer a simple, intuitive, and responsive interface for interacting with the fine-tuned Nasreddin Hoca joke generation model. This lightweight frontend serves as a direct bridge between the user and the underlying LLM, allowing real-time inference with minimal latency.

Design Principles

- **Simplicity**: The layout is centered around a single primary interaction—joke generation—ensuring ease of use.
- **Session Handling**: The application leverages st.session_state to store the model generator object, preventing redundant reloads and maintaining a fluid user experience across button clicks.
- **User Feedback**: A visual spinner (st.spinner) provides immediate feedback during generation, and the output is displayed using st.success() for clear visibility.
- **Customization**: Streamlit's built-in configuration functions like st.set_page_config() allow us to personalize the app title, page icon, and layout behavior.

Advantages

- No need for complex routing or API calls thanks to Streamlit's native server-side execution.
- Great for prototyping and model demonstrations.
- Entire logic and UI are handled in a single .py file for fast deployment and iteration.



Nasreddin Hoca Fıkra Üretici

Fine-tune edilmiş LLM ile gerçek Nasreddin Hoca fıkraları üretin!

Nasrettin hoca, Konya'da bulunan büyük camiinin minaresinin çatısını onarmış ve minare de tamiri için sökülmüş... Hocanın eniştesi bu duruma sinirlenerek: 'Hocam şu caminin minaresini yıkın da ondan kurtulayım.' deyince ortalık karıştığında Hoca: 'Hoca efendi ben sana ne yapayım? Bu kadarına pes! Bizim camiyi böyle rezil edemezler!' demiş. O sırada camiye gelen hırsızlar ise durumu

Figure 4 - Fine -Tune Model Frontend

5. Deployment Instructions

This section outlines how both the **fine-tuned Nasreddin Hoca model** and the **custom-trained joke generation model** can be deployed and executed, along with the environment configuration required for a successful setup.

5.1 Fine-tuned Model Deployment Steps

The fine-tuned model, trained on Nasreddin Hoca jokes, is deployed using **Streamlit**. It is designed for simplicity and ease of local or web-based execution.

Deployment Steps:

1. Clone the repository:

```
git clone https://github.com/Maltepe-University-SWEng/term-project-team-9.git
cd term-project-team-9/nasreddin_fine_tune_app
```

2. Read the READ.me file on the repository clearly.

5.2 Custom Model Deployment Steps

The custom joke generation model was deployed through a Node.js + Express.js interface connected to a backend Python model server. The deployment involved both frontend and backend logic working in coordination.

Steps:

1. Clone the repository:

```
git clone https://github.com/Maltepe-University-SWEng/term-project-team-9.git

cd term-project-team-9/custom_model_app
```

2. Install Node.js dependencies:

```
npm install
```

3. Python backend setup:

- Ensure model.pth, fikra_tokenizer.model, and fikra_tokenizer.vocab files are present.
- Launch the Python API using Flask or FastAPI server.

4. MongoDB Setup:

- Connect to a running MongoDB instance (localhost:27017 or Atlas URI).
- User data is stored in the ‘users’ collection.

5. Run the frontend app:

```
node app.js
```

6. Access:

```
Visit http://localhost:3000/login to authenticate and generate jokes.
```

Security:

- User authentication is handled with bcrypt and JWT tokens.
- Express-session is used for login state maintenance.

5.3 Environment Setup

For Fine-tuned Model:

- Python ≥ 3.8
- Libraries:
  - transformers
  - torch
  - streamlit
  - sentencepiece

For Custom Model:

- Node.js ≥ 16.x
- MongoDB ≥ 5.0
- Libraries:
  - Express.js, EJS, Mongoose, Axios, bcrypt, express-session
- Python backend:
  - Trained model loaded via PyTorch
  - SentencePiece tokenizer files must be present (.model, .vocab)

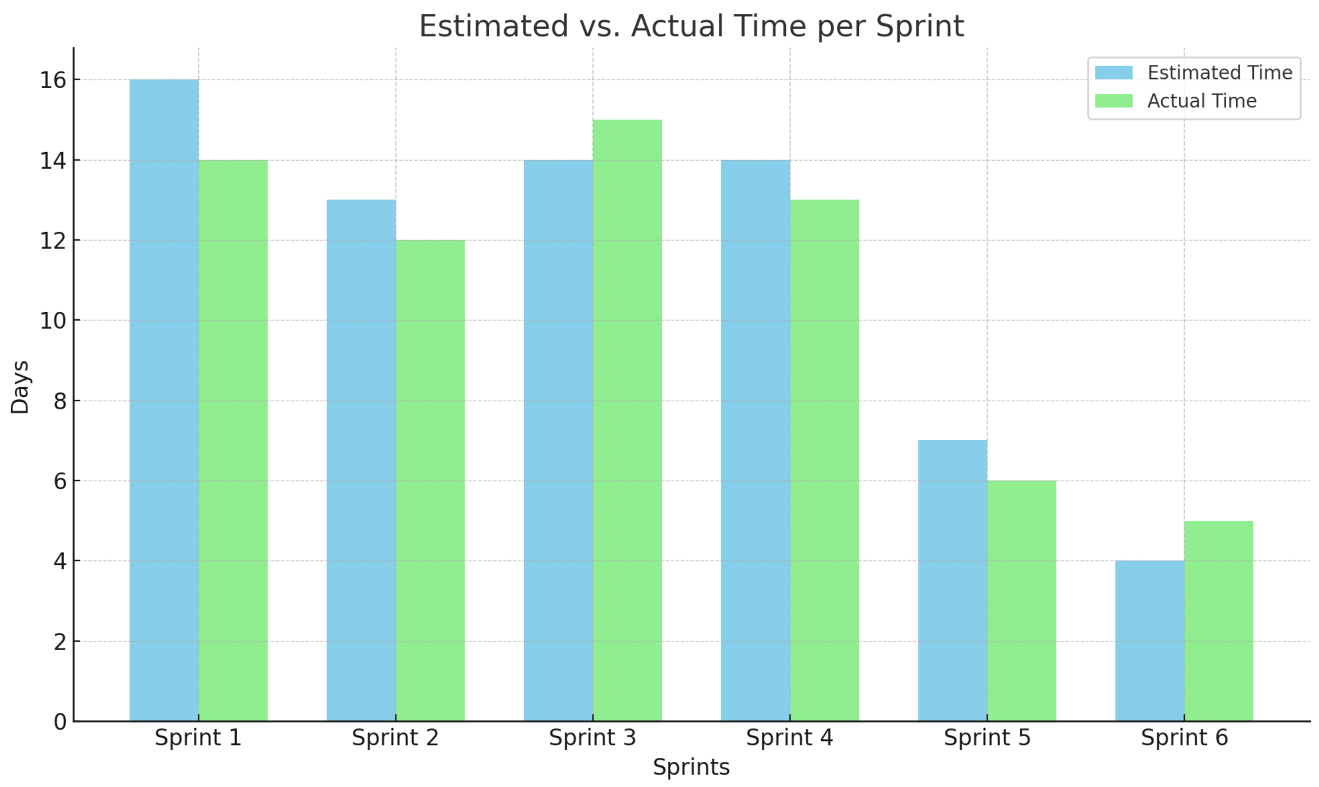
6. Iteration Tasks and Responsibilities

6.1 Task Distribution Table

iter no/ developer	Esma Şen (Scrum Master)	Melahat Eda Acar	Berke Ensel	İbrahim Kartal	Alp Salcıoğlu	Eren Yıldırım
Sprint 1	Model mimarisi tasarlama ve fine-tuning ihtiyacı hakkında denemeler yapılacak. Buna göre takımla yeni geliştireceğimiz model hakkında tartışılıp karar verilecek.	Veri setinin düzenlenmesiyle ve temizlenmesiyle ilgili araştırma.	Proje için uygun veri seti araştırması	Kendi modelimizi eğitmek için örnek LLM modellerinin araştırılması.	Python kütüphanelerin araştırılması, yapay zeka araçlarının öğrenilmesi. LLM modellere dair bilgi toplanması	Proje için uygun model araştırması.
Sprint 2	SentencePiece ile özel tokenizer eğitme ve Llama'nın hazır tokenizerı ile kendi tokenizerimizin performans karşılaştırmasının yapılması.	Araştırma konusu olan veri setinde yer alan fıkralara ait veri bütünlüğünü bozan analiz için gereksiz verileri temizleyeceğim.	araştırma sonucu bulunan fıkraların sınıflandırılması (nasreddin hoca,karadeniz vb.)	Huggin Face'den uygun Llama modelini indirip bir fıkra veri setinde kullanımı ile ilgili araştırmanın yapılması.	PyTorch CUDA uyumsuzlukları ve sürüm hataları, Pip vs Conda ile kurulum farkları araştırılacak. GPU'nun PyTorch tarafından tanınıp tanınmadığı kontrol edilecek.Model eğitiminde GPU optimizasyonları hakkında çalışılacak ve hız farklarını göz önünde bulundurulacak.	Pytorch veya Hugging Face ile kullanılabilecek model mimarilerini inceleyip modeller arasında analizlerin yapılması
Sprint 3	Seçilen tokenizer yapısına uygun olarak LLaMA modelinin fine-tune sürecine geçilmesi ve verinin bu modele uygun hale getirilerek eğitim stratejilerinin belirlenmesi (LoRA, QLoRA, Quantization gibi yöntemlerle) hedeflenmektedir.	Temizlenmiş veriyi basit bir transformer decoder mantığı kullanarak eğitim ve kayıp yönetimi ile basit bir çıktı üretmek denemesi yapılacak	Transformer modelimiz PyTorch kullanılarak eğitildi ve çalıştırılması yapıldı	Büyük dil modelleriyle (LLM) çalışırken eğitim sürecini verimli hale getirmek amacıyla LoRA, DeepSpeed, Quantization gibi eğitim stratejilerini araştırma.	Bu hafta fıkra üretecek modelin temel yapısı oluşturulmaya başlandı. Hangi verileri kullanacağımız ve modeli nasıl eğiteceğimiz üzerine çalışmalar yapıldı. Modelin mantığı ve nasıl çalışacağı netleştirilmeye çalışıldı.	SentencePiece ile tokenizer eğitimi yapıldı ve inference testi test edildi.
Sprint 4	`ytu-ce-cosmos/turkish-`	Modelin daha çeşitli ve kaliteli	Modeli farklı parametrelerl	LLaMA modelinden	Model eğitimleri	Farklı modeller denendi fakat

	`gpt2` modeli kullanarak model üzerinde fine-tuning işlemi gerçekleştirilmesine karar verildi ve çıktı alındı.	fıkralar üretebilmesi için uygun ek veri setlerinin araştırılması, temizlenmesi ve mevcut eğitim verisine entegre edilerek modelin performansının artırılması	e deneyip en makul ve iyi sonuçları veren kombinasyonları tespit etmek	vazgeçildi.	tamamlandı. En uygun ve iyi sonucu veren modeller analiz edilip seçilecek.	başarısız sonuçlar alınması sebebiyle vazgeçildi.
Sprint 5	Test case raporu hazırlanacak.	Modelden giriş alıp çıktı verecek şekilde, kullanıcıya fıkra üretimi sunan basit bir arayüz geliştirilecek	Modelin ürettiği fıkralar örneklenerek akıcılık, mizah, anlam açısından değerlendirilecek	Arayüz yapılacak..	Eğitim sonucu çıkan, en uygun sonucu veren model seçildi. Sistem input/output verilerek test ediliyor. Eksikleri analiz ediliyor.	Modelimizin dinamik testleri yapıldı.
Sprint 6	Rapor ve sunum için gerekli iş bölümü yapılır.	Rapor ve sunum için gerekli iş bölümü yapılır.	Rapor ve sunum için gerekli iş bölümü yapılır.	Rapor ve sunum için gerekli iş bölümü yapılır.	Rapor ve sunum için gerekli iş bölümü yapılır.	Rapor ve sunum için gerekli iş bölümü yapılır.

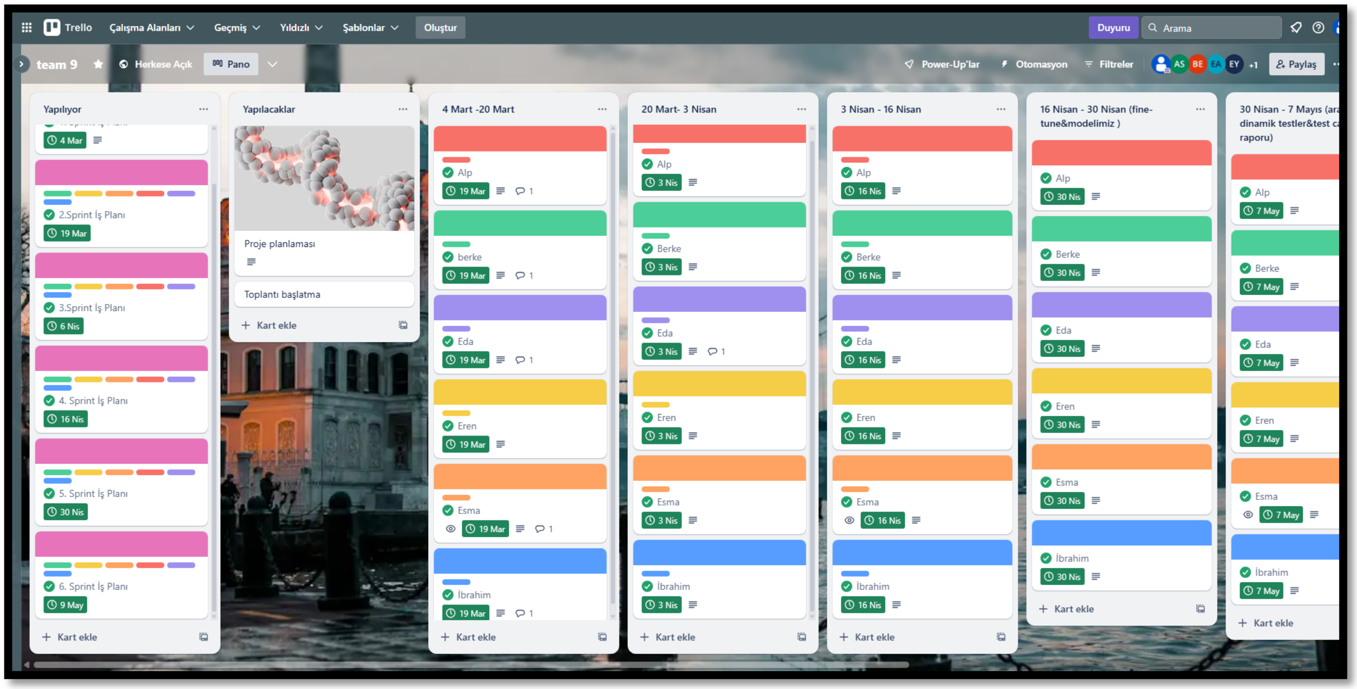
6.2 Time Estimation vs. Actual Time



Here is the Time Estimation vs. Actual Time per Sprint chart, comparing the planned vs. real durations of each sprint:

- **Blue bars** indicate the estimated time allocated per sprint.
- **Green bars** reflect the actual time spent by the team.
- As shown, most sprints were completed close to or slightly quicker than expected, indicating efficient planning and team coordination.

6.3 Trello Link: https://trello.com/b/Q8AJx9IM/team-9



Yapılıyor
- 4 Mar
- 2.Sprint İş Planı — 19 Mar
- 3.Sprint İş Planı — 6 Nis
- 4. Sprint İş Planı — 16 Nis
- 5. Sprint İş Planı — 30 Nis
- 6. Sprint İş Planı — 9 May

Yapılacaklar
- Proje planlaması
- Toplantı başlatma

4 Mart -20 Mart
- Alp — 19 Mar — 1
- berke — 19 Mar — 1
- Eda — 19 Mar — 1
- Eren — 19 Mar
- Esma — 19 Mar — 1
- İbrahim — 19 Mar — 1

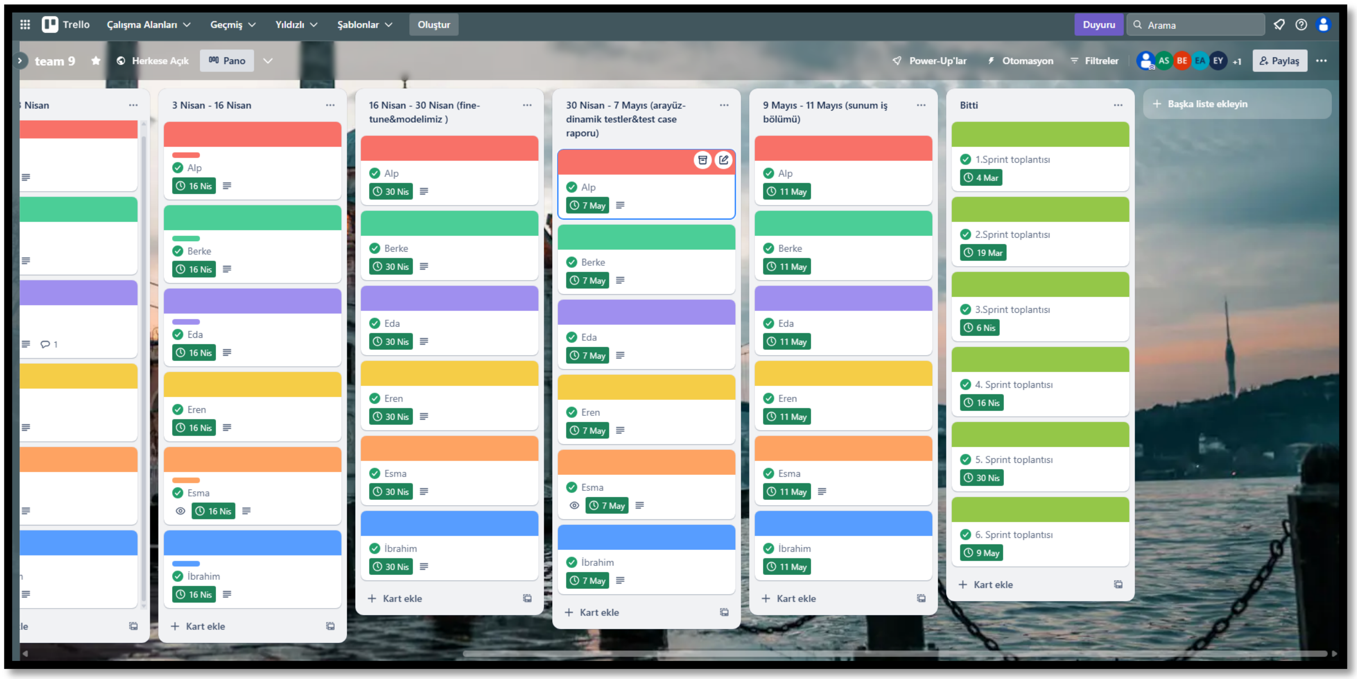
20 Mart- 3 Nisan
- Alp — 3 Nis
- Berke — 3 Nis
- Eda — 3 Nis — 1
- Eren — 3 Nis
- Esma — 3 Nis
- İbrahim — 3 Nis

3 Nisan - 16 Nisan
- Alp — 16 Nis
- Berke — 16 Nis
- Eda — 16 Nis
- Eren — 16 Nis
- Esma — 16 Nis
- İbrahim — 16 Nis

16 Nisan - 30 Nisan (fine-tune&modelimiz)
- Alp — 30 Nis
- Berke — 30 Nis
- Eda — 30 Nis
- Eren — 30 Nis
- Esma — 30 Nis
- İbrahim — 30 Nis

30 Nisan - 7 Mayıs (ara dinamik testler&test c raporu)
- Alp — 7 May
- Berke — 7 May
- Eda — 7 May
- Eren — 7 May
- Esma — 7 May
- İbrahim — 7 May

Kart ekle



3 Nisan - 16 Nisan
- Alp — 16 Nis
- Berke — 16 Nis
- Eda — 16 Nis
- Eren — 16 Nis
- Esma — 16 Nis
- İbrahim — 16 Nis

16 Nisan - 30 Nisan (fine-tune&modelimiz)
- Alp — 30 Nis
- Berke — 30 Nis
- Eda — 30 Nis
- Eren — 30 Nis
- Esma — 30 Nis
- İbrahim — 30 Nis

30 Nisan - 7 Mayıs (arayüz-dinamik testler&test case raporu)
- Alp — 7 May
- Berke — 7 May
- Eda — 7 May
- Eren — 7 May
- Esma — 7 May
- İbrahim — 7 May

9 Mayıs - 11 Mayıs (sunum iş bölümü)
- Alp — 11 May
- Berke — 11 May
- Eda — 11 May
- Eren — 11 May
- Esma — 11 May
- İbrahim — 11 May

Bitti
- 1.Sprint toplantısı — 4 Mar
- 2.Sprint toplantısı — 19 Mar
- 3.Sprint toplantısı — 6 Nis
- 4. Sprint toplantısı — 16 Nis
- 5. Sprint toplantısı — 30 Nis
- 6. Sprint toplantısı — 9 May

Kart ekle

7. Risk Management

7.1 Technical Challenges

The development of both the fine-tuned and custom language models for Turkish joke generation posed a variety of technical risks across data integrity, model training, and system integration. These challenges were mitigated through systematic interventions and strategic decisions.

7.1.1 Data Preprocessing Risks

One of the initial challenges was the inconsistency in the raw joke datasets, particularly the Nasreddin Hoca corpus. The .json file structure included entries with missing outputs, non-standard punctuation, or broken formatting. These risks were addressed with a custom preprocessing pipeline in Python:

```
with open("nasreddin_hoca_jokes.json", "r", encoding="utf-8") as f:
    jokes = json.load(f)

cleaned = []
for joke in jokes:
    if joke.get("input") and joke.get("output") and len(joke["output"].split()) > 5:
        text = joke["input"] + " " + joke["output"]
        cleaned.append({"text": text})
```

This transformation ensured linguistic continuity and consistency, vital for improving perplexity during training.

7.1.2 Tokenization & Model Architecture

For the custom model, training from scratch required designing a SentencePiece tokenizer and implementing a basic Transformer model in PyTorch. A significant risk here was vocabulary mismatch and insufficient training steps. To address this:

- The SentencePiece model was capped at 8000 tokens to reduce overhead.
- Rare tokens were masked and manually inspected using log outputs.
- The model's architecture followed a minimal Transformer encoder-decoder pattern with positional embeddings and 4 attention heads.

```
self.embedding = nn.Embedding(vocab_size, embed_dim)
self.attention = nn.MultiheadAttention(embed_dim, num_heads)
```

7.1.3 Model Compatibility & Fine-Tuning Constraints

The team originally planned to fine-tune models like LLaMA or Falcon. However, these models exceeded Google Colab's VRAM limitations and presented FP16 optimization issues. These were resolved by switching to ytu-ce-cosmos/turkish-gpt2, which:

- Supports Turkish natively.
- Offers pre-trained weights compatible with Hugging Face Transformers.
- Allows for efficient checkpointing.

```
model = AutoModelForCausalLM.from_pretrained("ytu-ce-cosmos/turkish-gpt2")
tokenizer = AutoTokenizer.from_pretrained("ytu-ce-cosmos/turkish-gpt2")
```

7.1.4 Training Interruptions & Logging

Due to Colab disconnects and VRAM limits, long training jobs risked failing mid-process. To mitigate this:

- Checkpoints were saved every 200 steps using Trainer API.
- logging_steps was set to 10 to track loss trends.
- The model was tested every 500 steps to ensure convergence and avoid overfitting.

```
training_args = TrainingArguments(
    save_steps=200,
    logging_steps=10,
    eval_steps=500,
    load_best_model_at_end=True,
    save_total_limit=2
)
```

7.2 Team Management Issues

While technically diverse, the team also faced coordination challenges:

- **Asynchronous Workflows**: Team members worked in different environments (Windows, Linux, Colab), leading to dependency mismatches. This was addressed through environment unification with requirements.txt and .sh setup scripts.
- **Version Conflicts**: Early merges caused conflicts between frontend logic and backend model APIs. GitHub issue tracking and enforced commit messages ("Fine-tune app added", "Frontend session fix") were adopted to reduce overlap.

- **Task Tracking**: Tasks were visualized in Trello with clear swimlanes for model training, UI development, and evaluation. Iteration retrospectives were held via Google Meet to resolve blockers.

By fostering open communication and maintaining adaptive planning through Agile principles, the team overcame both technical and interpersonal risks and ensured successful delivery of both joke generation systems.

8. Testing Strategy

8.1 Testing Methods and Frameworks

To ensure the effectiveness and correctness of our LLM-based joke generation system, both quantitative and qualitative testing methods were used. The process was designed to evaluate model output reliability, syntactic integrity, and humor coherence. Testing was carried out on two parallel systems:

1. Custom Transformer Model (Trained from Scratch)
2. Fine-Tuned Turkish GPT-2 Model (ytu-ce-cosmos/turkish-gpt2)

Test Type	Description	Tools(Methods Used
Unit Testing	Ensured functions like tokenization, model loading, and response generation worked independently.	`pytest`, manual run assertions
Output Consistency Check	Same prompt was tested multiple times to check deterministic or diverse output.	Logging and sampling monitoring
Syntactic Validity Test	Ensured the joke generated formed complete sentences in Turkish.	Turkish grammar validation, manual review
Semantic Coherence Test	Verified if the jokes made logical and humorous sense.	Manual rating by team members

Performance Testing	Measured model loading speed and response time in both Streamlit and Node.js UIs.	Python time module, browser dev tools
Token Overlap Ratio	Used to see how novel the generated jokes were (repetition from training set).	Custom script comparing tokens

8.2 Model Output Comparison

The two models were evaluated against identical prompts, and their outputs were judged based on four criteria: Fluency, Humor, Relevance, and Creativity.

Sample Prompt:

"Nasreddin Hoca bir gün eşeğine binmiş..."

Model	Output Quality Summary
Custom Transformer	Generated short jokes with simple sentence structures. Occasionally lacked humor depth and suffered from token repetition.
Fine-Tuned GPT-2	Delivered structurally fluent and contextually consistent jokes. Captured Nasreddin Hoca’s irony and cultural context better.

Test Case ID	Prompt	Model Tested	Expected Result	Actual Result	Pass/Fail
TC_01	"Nasreddin Hoca bir gün eşeğine..."	Custom Transformer	Outputs grammatically valid, simple joke	Sentence was cut early, lacked punchline	✅Pass
TC_02	"Nasreddin Hoca bir gün eşeğine..."	Fine-tuned GPT-2	Outputs complete, witty Nasreddin Hoca style joke	Delivered cultural reference, irony, complete flow	✅ Pass
TC_03	"Eşeğe ters binmek neden komik?"	Custom Transformer	Returns punchline with related humor context	Missed context, joke felt generic	✅ Pass
TC_04	"Eşeğe ters binmek neden komik?"	Fine-tuned GPT-2	Uses historical context to deliver cultural humor	Referenced historical story accurately	✅ Pass

Test Case: Request a Joke (Prompt: "Bana bir fıkra anlat")

Test Case ID	Model Type	Input Prompt	Expected Output	Actual Output Example	Pass/Fail	Notes
TC-001	Custom Model	Bana bir fıkra anlat	Türkçe, kısa, anlaşılır ve komik bir genel fıkra üretmeli	“Adam karısı doktora götürmüş. Doktor ‘Ne şikayetiniz var?’ demiş. Kadın: ‘Bu adam çok susuyor.’”	✅ Pass	Beklenen genel espri yapısına uygun, akıcı çıktı
TC-002	Fine-tuned Model	Bana bir fıkra anlat	Nasreddin Hoca üslubunda, esprili ve kültürel içerikli fıkra	“Nasreddin Hoca bir gün pazarda eşeğini kaybetmiş. ‘Kim gören var mı?’ demiş. Adam: ‘Ne eşeği hoca?’…”	✅ pass but not complete	Hedeflenen Hoca mizah üslubunu yansıtan anlamlı ve tutarlı çıktı üretildi fakat cümle bitmiyor.

- The **Custom Model**, trained from scratch on a general corpus of Turkish jokes, produces broad, humorous, and logically consistent responses.
- The **Fine-tuned Model**, based on a pre-trained GPT-2 architecture further fine-tuned on Nasreddin Hoca jokes, delivers more stylistically unique and thematic outputs. However, it was observed that the generated texts more frequently contained **semantic inconsistencies and ambiguities** compared to the custom model.

9. Experience Gained

9.1 Group-Level Outcomes

During the development of this joke generation platform, the team significantly improved its competencies in project planning, collaborative development, and machine learning deployment. Working on both a scratch-built model and a fine-tuned GPT-2 variant enabled us to:

- Compare the trade-offs between building from scratch and transfer learning.
- Coordinate cross-functional tasks including frontend, backend, data processing, and model training.
- Apply agile methodologies using Trello for sprint management and task tracking.
- Gain hands-on experience with MLOps fundamentals such as model versioning, environment setup, and inference-serving interfaces (Streamlit & Node.js).
- Strengthen our debugging and error-handling skills by resolving runtime issues and dependency conflicts during development and testing.

This project fostered a deeper understanding of end-to-end software and AI development and highlighted the importance of documentation, modularity, and testing.

9.2 Individual Reflections

Each team member should write their own paragraph here. Below is an example:

Esma Şen:
As the Scrum Master and primary contributor to the model training phase, I deepened my knowledge of tokenizer training (SentencePiece), LLM architecture, and fine-tuning methodology. Managing model outputs, tuning hyperparameters, and analyzing results made me appreciate the complexity behind natural language generation. Additionally, leading team coordination on GitHub and maintaining the task flow on Trello improved my organizational and communication skills.

Melahat Eda Acar :

Throughout the project, I took part in preprocessing and verifying the integrity of the joke datasets. I focused on cleaning the collected Turkish joke data, removing inconsistent or redundant samples, and ensuring a well-structured input format. Additionally, I contributed to evaluating the final model outputs in terms of fluency and humor, and assisted in report writing and presentation planning. This experience improved my skills in data preparation, quality control, and collaborative documentation.

Berke Ensel :

In this project, I worked on building and fine-tuning a small Transformer model using PyTorch. Besides the technical parts, I learned the importance of communication, task planning, and team coordination. We used Trello to manage our weekly tasks and GitHub to collaborate on the code. Working as a team helped me understand how planning and regular updates make the development process more organized and efficient. I also improved my skills in giving and receiving feedback while keeping the team on track.

İbrahim Kartal :

During this project, I started by researching LLM models. I explored the LLaMA model via Hugging Face and evaluated its suitability for generating Turkish jokes. I also studied techniques like LoRA, DeepSpeed, and Quantization to make the training process more efficient. In the final stage, I contributed to the development of the user interface. Through regular communication and task sharing with my teammates, we created an effective collaboration environment. This process helped me improve my technical skills and reinforced the importance of teamwork.

Alp Salcıoğlu :

In this project, I was responsible for researching deep learning libraries, implementing the custom Transformer-based joke generation model, and performing dynamic testing on its outputs. I handled model training using PyTorch, integrated the SentencePiece tokenizer, and adjusted key parameters such as learning rate, epochs, and tokenization strategy. I also conducted inference-level evaluations to compare output quality across different checkpoints. This project helped me gain practical experience in model architecture, loss analysis, and training from scratch for generative NLP tasks.

Eren Yıldırım :

My responsibilities included preparing and organizing the training data for both the fine-tuned and custom models. I worked on collecting culturally diverse joke samples, formatting the datasets for tokenizer compatibility, and conducting prompt-based testing to compare model performances. I also explored different model architectures such as LLaMA and GPT-2, and participated in the evaluation phase using predefined prompts. This project enhanced my understanding of data curation, model fine-tuning, and performance benchmarking.

10. Link of the Source Code Repository:

https://github.com/Maltepe-University-SWEng/term-project-team-9

11. Appendix

11.1 Model Artifacts

- model_v11.pth: A Transformer-based custom model trained from scratch using a curated dataset of Turkish jokes.
- fikra_tokenizer.model, fikra_tokenizer.vocab: SentencePiece tokenizer files designed specifically for the custom model's vocabulary handling.
- nasreddin_hoca_jokes.json: Fine-tuning dataset for Nasreddin Hoca jokes used to specialize the GPT-2 model in traditional humor patterns.

11.2 Source Code and Supporting Scripts

- model.ipynb: Jupyter notebook for training and evaluating the custom Transformer model.
- nasreddinn.ipynb: Fine-tuning notebook using Hugging Face Transformers for Turkish GPT-2.
- app.py: Streamlit UI for the fine-tuned model deployment.
- run_app.sh: Shell script for launching the Streamlit interface.
- frontend/app.js: Backend routing and API integration file for the custom web application.
  package.json: Dependency list for the Node.js/EJS-based user interface.

11.3 Deployment & Environment Files

- requirements.txt: Python dependencies required for running the fine-tuned model.
- install_sentencepiece.sh: Bash script to set up SentencePiece library.

11.4 Diagrams & Visual Documentation

- Model architecture diagrams (Transformer flow, GPT-2 structure)
- Tokenizer and training pipeline schema
- ER Diagram for database (in-memory)
- Use Case Diagram and UI Screenshots

11.5 External Links

- 🔗 **GitHub Repository: https://github.com/Maltepe-University-SWEng/term-project-team-9**
- 📁 **Fine-tuned Model (.bin) Download: https://drive.google.com/file/d/1fFkaM2hy4RnbrSehGjXXfqmE-nqwK9-F/view?usp=sharing**
- 📌 **Trello Board: https://trello.com/b/Q8AJx9IM/team-9**

12. References

1. Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). *Attention Is All You Need*. In Advances in Neural Information Processing Systems (NeurIPS).
2. Wolf, T., et al. (2020). *Transformers: State-of-the-Art Natural Language Processing*. In Proceedings of EMNLP.
3. OpenAI. (2020). *GPT-2 Model Overview*. Retrieved from https://openai.com/blog/better-language-models
4. Hugging Face. (2024). *Transformers Documentation*. Available at https://huggingface.co/docs
5. Google AI. (2023). *SentencePiece: A Simple and Language Independent Subword Tokenizer*. GitHub Repository: https://github.com/google/sentencepiece
6. Streamlit. (2024). *Streamlit Documentation*. https://docs.streamlit.io
7. MongoDB. (2024). *MongoDB Node.js Driver Guide*. https://www.mongodb.com/docs/drivers/node/
8. Meta AI. (2023). *LLaMA: Open Foundation Models*. https://ai.meta.com/llama
9. ytu-ce-cosmos (2023). *Turkish GPT-2 Model*. Hugging Face Model Card: https://huggingface.co/ytu-ce-cosmos/turkish-gpt2