MALTEPE UNIVERSITY

TURKISH JOKE GENERATOR LLM

SE 403

SOFTWARE PROJECT MANAGEMENT

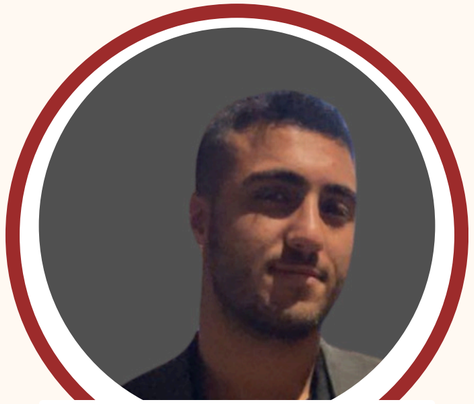
MEET WITH GROUP 7

ÖMER FARUK ÖZER
SCRUM MASTER/MODEL DEVELOPER

FURKAN AKSOY
LEAD DEVELOPER

EMRE SARI
FRONT-END DEVELOPER

YAREN YILDIZ
DATA SCIENTIST



MEHMET GÜZEL
BACK-END DEVELOPER

TAMAY YAZGAN
QA TESTER

PROJECT INTRODUCTION

You can compare the two different artificial intelligence models used in our project and examine the results they produce for the same input.

GPT-2 Model

Transformer-Based Language Model

The GPT-2 based language model, specially trained using the Hugging Face Transformers library, has the ability to understand comprehensive texts and produce coherent jokes in context.

- Can produce longer and more complex jokes
- Can better understand context
- Flexible generation across various joke types

LSTM Model

Recurrent Neural Network Model

The LSTM-based language model developed by our team using PyTorch offers a lighter and faster artificial intelligence solution.

- Faster joke generation
- Lower resource consumption
- Higher success with repetitive patterns

Our system allows you to compare the quality and originality of jokes produced by both models. This way, you can see how artificial intelligence models interpret Turkish humor and which model is more successful in which types of jokes.

GROUP7

TECHNOLOGIES

GPT-2

The GPT-2 model was fine-tuned with our dataset.

Back-end was developed using Flask and model training with Python.

LSTM

An LSTM model was developed from scratch.

HTML

A website skeleton was created using HTML.

Bootstrap was used for effects and enhancements.

CSS

Website designs were created using CSS.

GROUP7

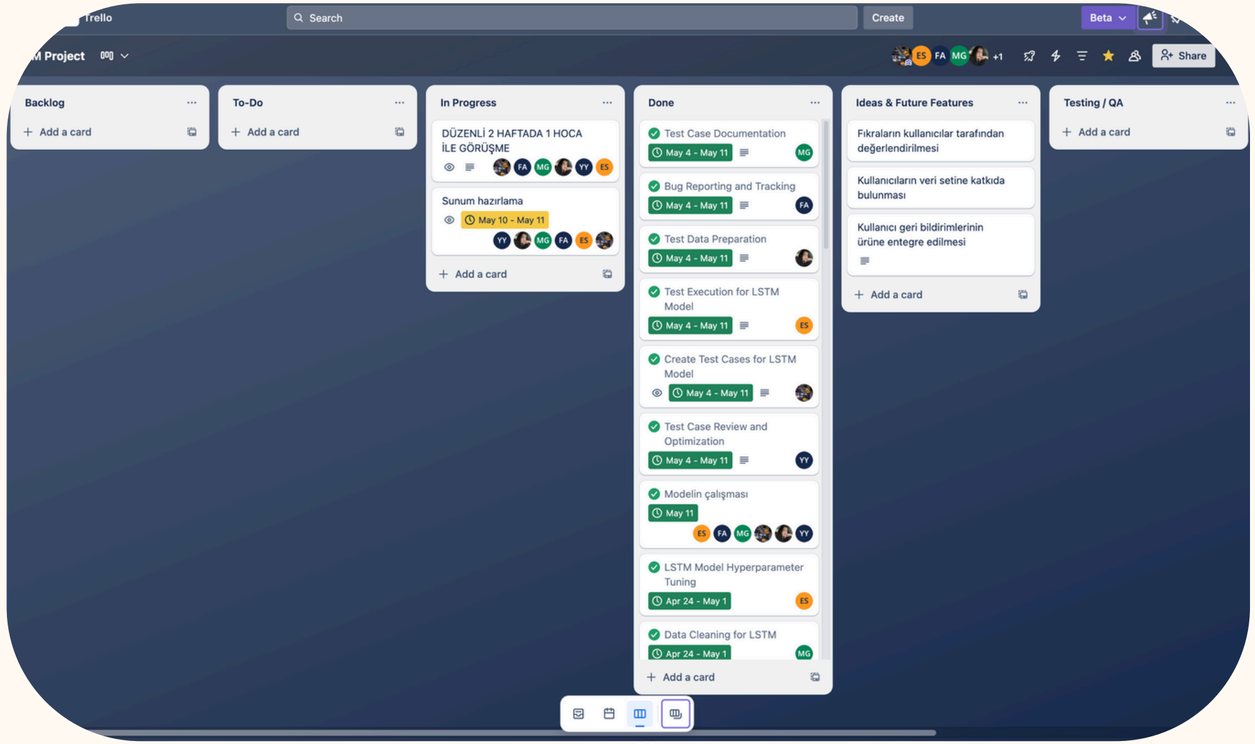
CHALLENGES

The technical challenges we faced were significant:

- Building a comprehensive dataset of Turkish jokes
- Fine-tuning language models for Turkish content
- Developing evaluation metrics for humor quality
- Creating a responsive web interface for interacting with our models

GROUP7

PROCESS



Trello

Person	Task	Sprint	Effort	Duration	Start	
er Faruk Özer	Model Architecture Review	Sprint 1	11.0	3	2025-02-20	2025-02
Yaren Yıldız	Advanced Model Comparison	Sprint 1	16.0	4	2025-02-23	2025-02-2
Tamay Yazgan	Prompt Patterns Analysis	Sprint 1	16.0	4	2025-02-27	2025-03-02
Tamay Yazgan	Dataset Scope Evaluation	Sprint 1	8.0	2	2025-03-03	2025-03-04
Furkan Aksoy	Basic Code Architecture	Sprint 1	16.0	4	2025-03-05	2025-03-08
Mehmet Güzel	API Call Research	Sprint 1	9.0	2	2025-03-09	2025-03-10
Emre Sarı	Dataset Source Collection	Sprint 1	8.0	2	2025-03-11	2025-03-12
Ömer Faruk Özer	Code Cleanup and Refactoring	Sprint 2	11.0	3	2025-03-13	2025-03-15
Tamay Yazgan	Unit Test Creation	Sprint 2	10.0	3	2025-03-16	2025-03-18
Mehmet Güzel	Backend Framework Comparison	Sprint 2	16.0	4	2025-03-19	2025-03-22
Furkan Aksoy	Prompt Test Scenarios	Sprint 2	17.0	4	2025-03-23	2025-03-26
Emre Sarı	Model Performance Reporting	Sprint 2	8.0	2	2025-03-27	2025-03-28
Yaren Yıldız	Database Management Systems Review	Sprint 2	19.0	5	2025-03-29	2025-04-02
Furkan Aksoy	Test Automation Review	Sprint 3	11.0	3	2025-04-03	2025-04-05
Yaren Yıldız	Comprehensive Data Preprocessing	Sprint 3	14.0	4	2025-04-06	2025-04-09
Mehmet Güzel	Documenting the Coding Process	Sprint 3	16.0	4	2025-04-10	2025-04-13
Emre Sarı	Test Report Preparation	Sprint 3	13.0	3	2025-04-14	2025-04-16
Ömer Faruk Özer	Backend Architecture Design	Sprint 3	15.0	4	2025-04-17	2025-04-20
Emre Sarı	Data Validation Techniques	Sprint 4	16.0	4	2025-04-21	2025-04-24
All Team	Backup Strategy Review	Sprint 4	25.0	6	2025-04-25	2025-04-30
Ömer Faruk Özer	LSTM Model Data Preparation	Sprint 5	12.0	5	2025-04-15	2025-04-19
Yaren Yıldız	LSTM Model Training	Sprint 5	12.0	5	2025-04-20	2025-04-24
Tamay Yazgan	LSTM Model Evaluation	Sprint 5	12.0	5	2025-04-25	2025-04-29
Furkan Aksoy	Data Collection for LSTM	Sprint 5	12.0	5	2025-04-30	2025-05-0
ehmet Güzel	Data Cleaning for LSTM	Sprint 5	12.0	3	2025-05-05	2025-05
e Sarı	LSTM Model Hyperparameter Tuning	Sprint 5	12.0	4	2025-05-04	2025

Effort And Duration Table

Throughout our development cycle, we implemented Scrum methodologies with two-week sprints, daily stand-ups, and regular retrospectives to continuously improve our approach. This enabled us to iterate quickly and address challenges as they arose.

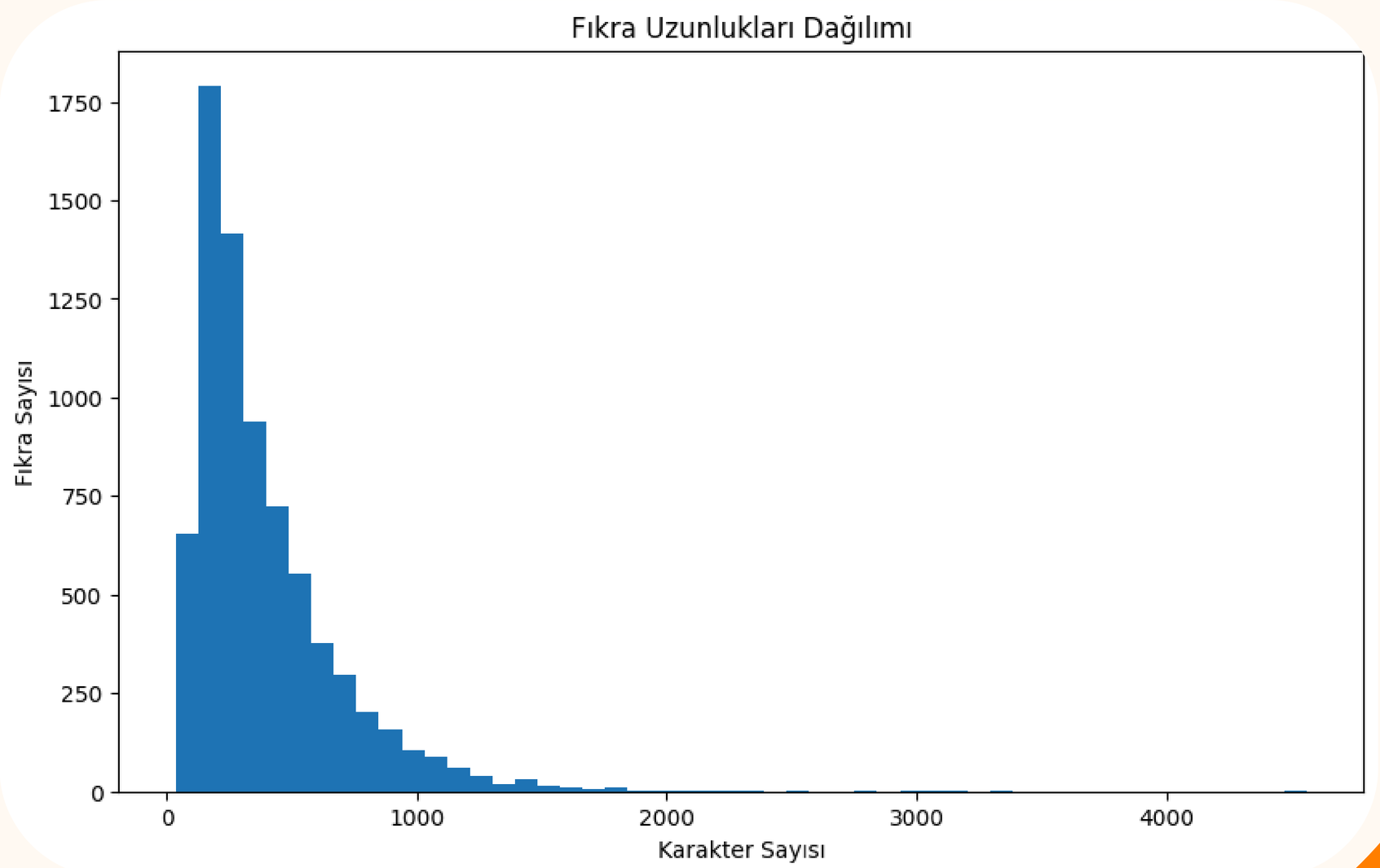
GROUP7

DATASET PREPARATION AND SIMILARITY ANALYSIS

I led the data collection and preprocessing efforts for this project.
To train effective joke generation models, we needed a substantial and diverse dataset. We compiled a corpus of 7,500 traditional Turkish jokes across three categories: Nasreddin Hoca, Temel, and general jokes.

GROUP7

JOKES LENGTH DISTRIBUTION



WEB SCRAPING


```python
import requests
from bs4 import BeautifulSoup

# URL of the webpage to scrape
url = "https://example.com"

# Fetch the webpage content using requests
response = requests.get(url)

# Parse the HTML content with BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')

# Find all heading tags (h1, h2, h3)
headers = soup.find_all(['h1', 'h2', 'h3'])

# Print the text content of each heading
for header in headers:
    print(header.text.strip())
```

Jokes were automatically collected from the internet as data using web scraping.

GROUP7

DATA COLLECTION

Our data collection methodology combined multiple approaches:

- Web scraping from cultural websites and joke archives
- Optical Character Recognition (OCR) on digitized joke books
- Manual transcription from printed joke collections
- Data augmentation techniques for underrepresented categories

The dataset was structured in JSON format with three key fields:

GROUP7

DATASET JSON FORMAT

We determined three headings for the dataset format: Kategori, Baslik and Metin.


```
37800     "kategori": "Genel",
37801     "baslik": "Öğretmen:\n- Oğlum, yerçekimini kim buldu",
37802     "metin": "Öğretmen:\n- Oğlum, yerçekimini kim buldu?\nÖğrenci:\n- Newton, ama itira
37803   },
37804   {
37805     "kategori": "Nasreddin",
37806     "baslik": "Öğrenci:\n- Hocam, Afrika'da kaç ülke var",
37807     "metin": "Öğrenci:\n- Hocam, Afrika'da kaç ülke var?\nÖğretmen:\n- 54.\nÖğrenci:\n-
37808   },
37809   {
37810     "kategori": "Temel",
37811     "baslik": "Doktor:\n- Sigara içme, alkol alma, spor yap",
37812     "metin": "Doktor:\n- Sigara içme, alkol alma, spor yap!\nTemel:\n- Doktor, bunları
37813   },
37814   {
37815     "kategori": "Genel",
37816     "baslik": "Hasta:\n- Doktor, röntgen çekilirken bağırmam ge...",
37817     "metin": "Hasta:\n- Doktor, röntgen çekilirken bağırmam gerekir mi?\nDoktor:\n- Hay
37818   },
37819   {
37820     "kategori": "Genel",
37821     "baslik": "Patron:\n- Dursun, seni işe alamayız, çok tembelsin",
37822     "metin": "Patron:\n- Dursun, seni işe alamayız, çok tembelsin!\nDursun:\n- Tembel d
37823   }
37824 ]
```

JOKE SIMILARITY CHECK & OVERFITTING PREVENTION

First, we load all jokes from our JSON files and clean them—removing punctuation, extra spaces, and converting to lowercase. For each newly generated joke, we compute a similarity ratio against every stored joke using Python's SequenceMatcher and record the highest score.

If the score exceeds 0.8, we flag it as a duplicate and discard it. Scores between 0.5 and 0.8 trigger a manual review. Anything below 0.5 we accept as novel. This check prevents the model from simply repeating training data, curbs overfitting, and ensures every joke our system produces is genuinely fresh.

CHECK DATASET

Terminal Local

```
(.venv) PS C:\Users\furka\Desktop\term-project-team-7> python check_joke.py -t "Nasreddin Hoca bir gün panayıra gitmiş"
fikralar.json bulundu  data\fikralar.json
data\fikralar.json dosyasından 7568 fıkra yüklendi.
data\nasreddin.json dosyasından 50 Nasreddin fıkrası yüklendi.
data\temel.json dosyasından 476 Temel fıkrası yüklendi.
Toplam 8046 fıkra yüklendi.

======= FIKRA KONTROL SONUÇLARI =======
Kontrol edilen fıkra. "Nasreddin Hoca bir gün panayıra gitmiş..."
Benzerlik oranı: 44.44%

❌BU FIKRA VERİ SETİNDE BULUNMUYOR
Bu muhtemelen model tarafından üretilmiş yeni bir fıkradır.

En yakın fıkra (düşük benzerlik):
"\"Bana 'sokak kedisi' deme, 'şehir müzesi canlı sergisi' de!\""
```

The script loads a total of 7,500 jokes from three JSON files and then computes a 44.44% similarity score for the input “Nasreddin Hoca bir gün panayıra gitmiş.” Since this is below the 50% threshold, it flags the joke as not found in the dataset and likely newly generated by the model.

GPT-2 MODEL TRAINING AND IMPLEMENTATION

For this project, we specifically selected and fine-tuned a GPT-2 architecture to work exclusively with Turkish joke generation. Unlike approaches that begin with English-pretrained models, we focused on optimizing our model entirely for Turkish language patterns and humor structures from the outset.

GROUP7

GPT-2 FOR TURKISH JOKE GENERATION

Our training was conducted exclusively on our specialized corpus of 7,500 Turkish jokes across three distinct categories. This approach ensured that the model developed a deep understanding of Turkish linguistic patterns, cultural references, and the unique structural elements of different joke types within Turkish humor tradition.

GPT-2

GROUP7

GPT2 MODEL TRAINING

For the technical specifications, we used:

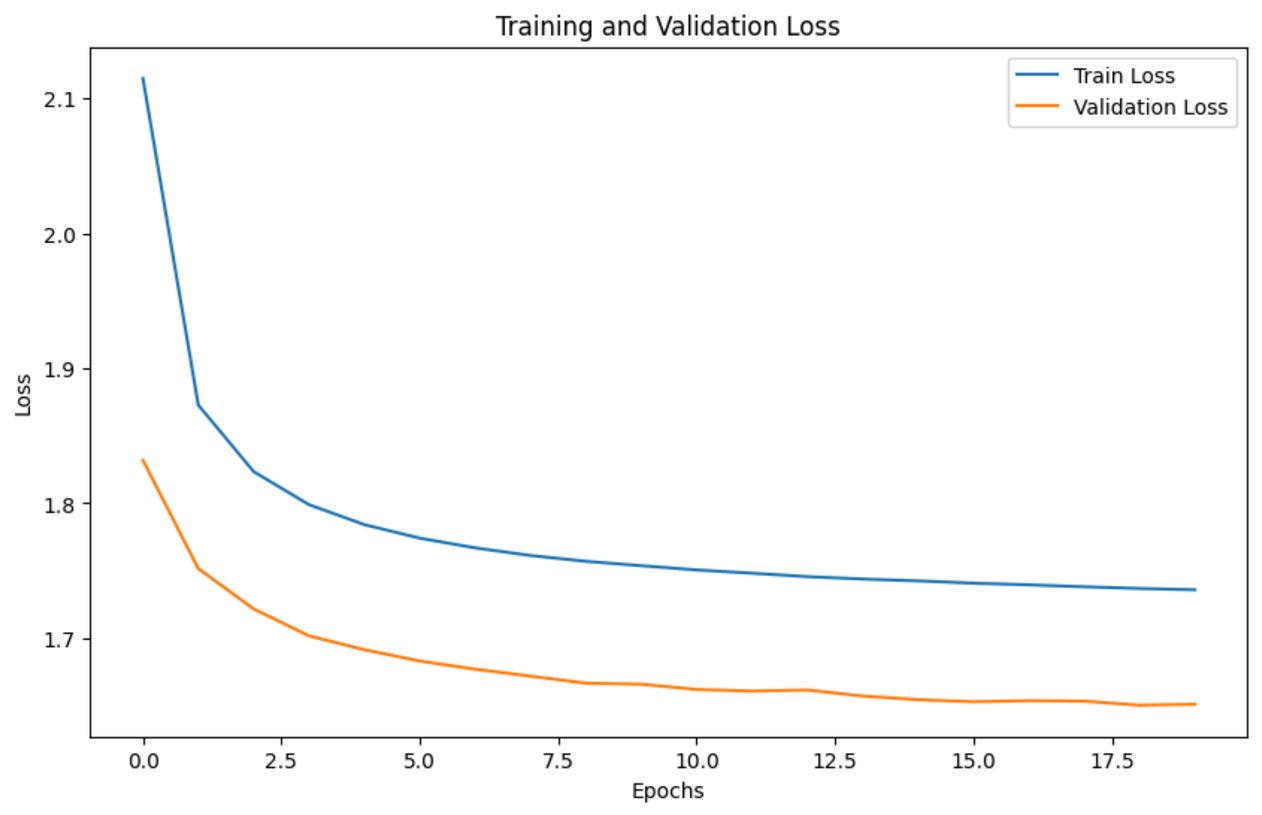
Model: GPT-2 medium (355M parameters)

Training infrastructure: Google Colab with GPU acceleration

Training duration: Approximately 30 hours
Number of epochs: 25

Checkpoints: Saved every 5 epochs to prevent work loss

Final training loss: Reduced to 1.36



FINE-TUNING APPLICATION - 1

```python
# Fıkra kalitesini değerlendiren fonksiyon
def score_joke(joke_text):  5 usages  new *
    """Fıkranın kalitesini değerlendiren skorlama fonksiyonu"""
    score = 0

    # Uzunluk puanı - daha uzun fıkralar genellikle daha iyidir (belli bir limite kadar)
    words = joke_text.split()
    if len(words) < 10:
        score -= 10  # Çok kısa fıkralar kötüdür
    elif len(words) > 60:
        score += 15  # Uzun fıkralar iyidir
    else:
        score += len(words) / 4  # Orta uzunluktaki fıkralar için orantılı puan

    # Dilbilgisi puanı   cümle sonu noktalama işaretleri
    if joke_text.endswith(('.', '!', '?')):
        score += 5  # Düzgün biten fıkralar daha iyi
```

One innovative aspect of our implementation is the multi-attempt generation strategy. If the first attempt doesn't produce a high-quality joke (based on our scoring algorithm), the system automatically tries again with different parameters, selecting the highest-scoring result from multiple generations.

GROUP7

FINE-TUNING APPLICATION - 2

```
# Tutarlılık puanı   başında "iki" kelimesi var mı (model sorunuydu)
if joke_text.lower().startswith("iki"):
    score -= 5  # "iki" ile başlayan fıkralardan kaçın

# Dialog puanı - konuşma içeren fıkralar genellikle daha iyidir
if '-' in joke_text or '"' in joke_text:
    score += 10

# Karakter puanı   popüler karakterleri içeriyor mu?
if "Temel" in joke_text or "Nasreddin" in joke_text or "Hoca" in joke_text:
    score += 8

# Anlam puanı   saçma sapan kelime dizileri içermiyor mu?
nonsense_patterns = ["nann"  "ÇALAR", "ĞANIN"  "BURADA K", "GEL"]
for pattern in nonsense_patterns:
    if pattern in joke_text:
        score -= 8

return score
```

We also implemented post-processing functions that:

- Clean up the generated text
- Fix capitalization
- Remove any nonsense patterns or repeated segments
- Ensure proper punctuation and sentence structure

GROUP7

FINE TUNENING GPT2 MODEL - 1

One of the most significant aspects of our implementation, as you can see in our app.py code, is the joke generation pipeline we developed. This includes:

```python
# GPT-2 modeli ile fıkra üretme
def generate_gpt2_joke(joke_type="random", max_attempts=3):  2 usages  FurkanAksoyy *
    global global_gpt2_model, global_gpt2_tokenizer

    # Model yüklü değilse hata döndür
    if global_gpt2_model is None or global_gpt2_tokenizer is None:
        print("GPT-2 modeli yüklenemedi")
        return None, None, False
```

```python
# Farklı üretim parametreleri
param_sets = [
    # Tutarlı parametreler
    {"temperature": 0.7, "top_p": 0.92, "top_k": 50, "repetition_penalty": 1.2, "max_length": 200},
    # Yaratıcı parametreler
    {"temperature": 0.8, "top_p": 0.95, "top_k": 60, "repetition_penalty": 1.1, "max_length": 220},
    # Dengeli parametreler
    {"temperature": 0.75, "top_p": 0.9, "top_k": 55, "repetition_penalty": 1.15, "max_length": 210}
]
```

FINE TUNENING GPT2 MODEL - 2

We implemented several specialized techniques

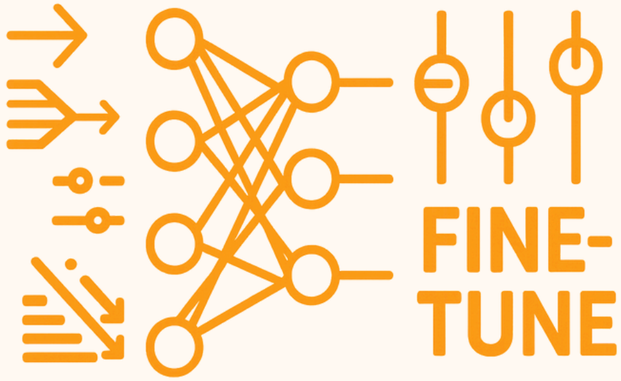
- Custom prompt templates to properly format jokes by category
- Multiple parameter configurations to optimize joke quality
- A sophisticated joke scoring system that evaluates:
- Length and structure appropriateness
- Grammar and punctuation quality
- Character recognition (Nasreddin Hoca, Temel)
- Dialogue presence (which improves joke quality)
- Avoidance of nonsensical patterns

GROUP7

OUTPUTS OF GPT2 FINE TUNENING

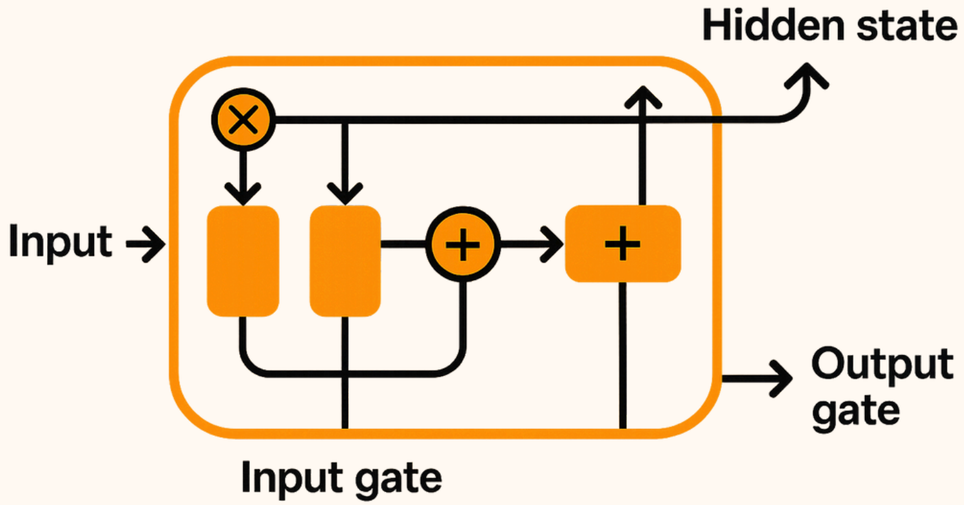
This scoring system allows us to programmatically evaluate joke quality, helping us select the best outputs from multiple generation attempts.

The final trained model showed impressive abilities to capture the unique patterns of Turkish joke construction, including the specific narrative styles and punchline structures characteristic of each category.



LSTM (Long Short-Term Memory) Neural Network Structure

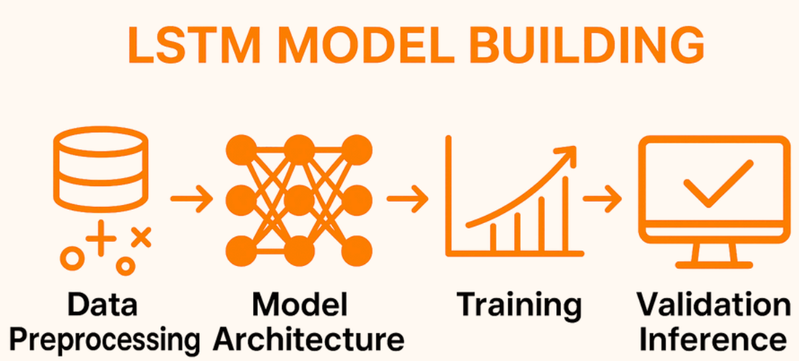
As a complementary approach to our GPT-2 implementation, we developed a character-level LSTM model for Turkish joke generation. LSTM (Long Short-Term Memory) networks excel at capturing sequential patterns with their specialized memory cells that can retain information over long distances in text. We specifically chose a character-level approach rather than word-level because Turkish is an agglutinative language with complex morphology, allowing our model to learn character combinations that form authentic Turkish linguistic patterns even when generating novel content.



LSTM MODEL BUILDING

Let me explain the architecture of our LSTM model:
We implemented a multi-layer character-level LSTM with the following specifications:

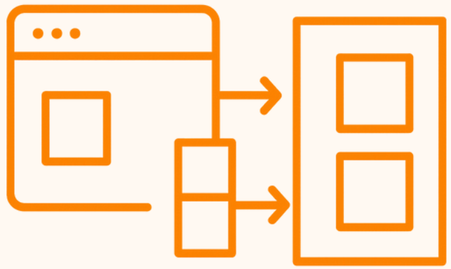
- Input: One-hot encoded characters (vocabulary size of 126, covering all Turkish characters plus special tokens)
- Embedding layer: 256 dimensions
- Hidden layers: 3 LSTM layers with 512 hidden units each
- Dropout: 0.5 between layers for regularization
- Output: Fully connected layer projecting to character probabilities
- Activation: Softmax function for character prediction



GROUP7

STRUCTURE

```
class JokeLSTM(torch.nn.Module):  3 usages  new *
    def __init__(self, vocab_size, embedding_dim=256, hidden_dim=512, num_layers=3, dropout=0.5):  new *
        super(JokeLSTM, self).__init__()
        self.embedding = torch.nn.Embedding(vocab_size, embedding_dim)
        self.lstm = torch.nn.LSTM(
            input_size=embedding_dim,
            hidden_size=hidden_dim,
            num_layers=num_layers,
            dropout=dropout if num_layers > 1 else 0,
            batch_first=True
        )
        self.dropout = torch.nn.Dropout(dropout)
        self.fc = torch.nn.Linear(hidden_dim, vocab_size)
```

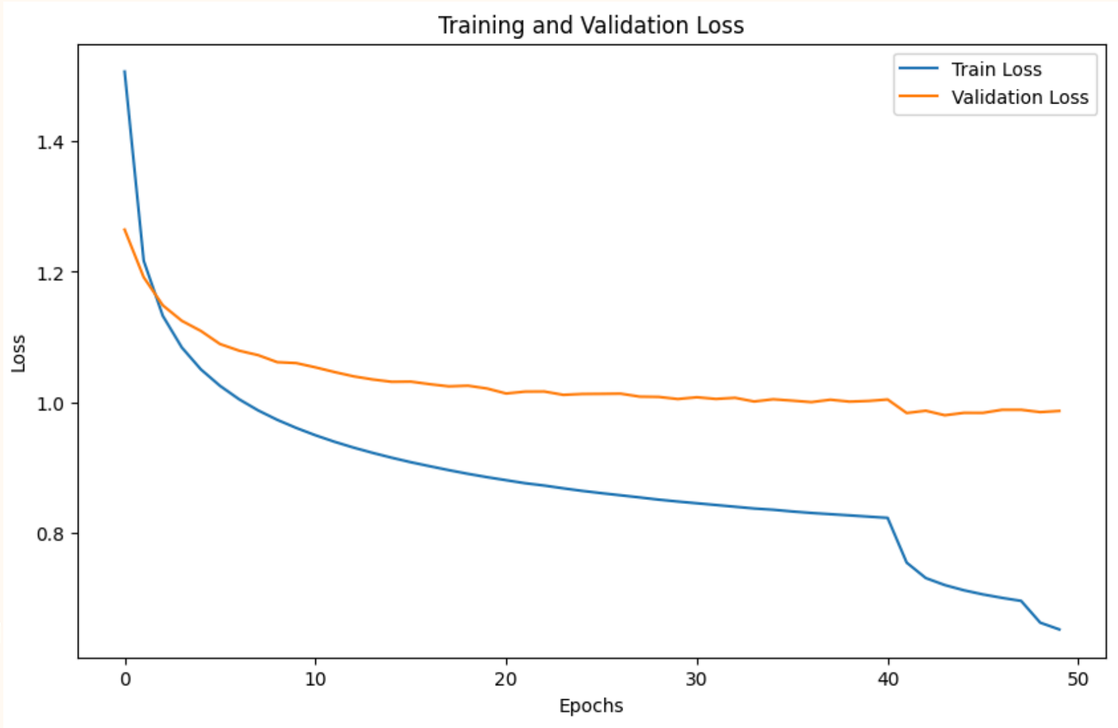


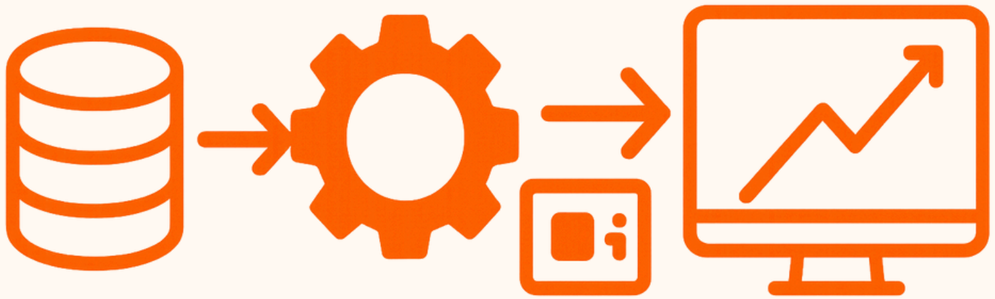
GROUP7

LSTM MODEL TRAINING

Training process details:

- Platform: Google Colab with GPU acceleration
- Training duration: 20 hours across 50 epochs
- Batch size: 128 sequences
- Sequence length: 500 characters
- Optimizer: Adam with learning rate of 0.001
- Loss function: Cross-entropy
- Character-level accuracy reached 65.4% by the final epoch





ENSURING JOKE QUALİTY

We used several special techniques to improve joke generation quality:

- Category-specific start tokens (<NASREDDIN>, <TEMEL>, <GENEL>)
- Temperature sampling for controlling randomness during generation
- Hidden state initialization strategies to guide the narrative flow
- Custom stopping criteria based on sequence coherence

COMPARISION OF OUR MODELS

The character-level approach offers some advantages compared to the token-level GPT-2 model:

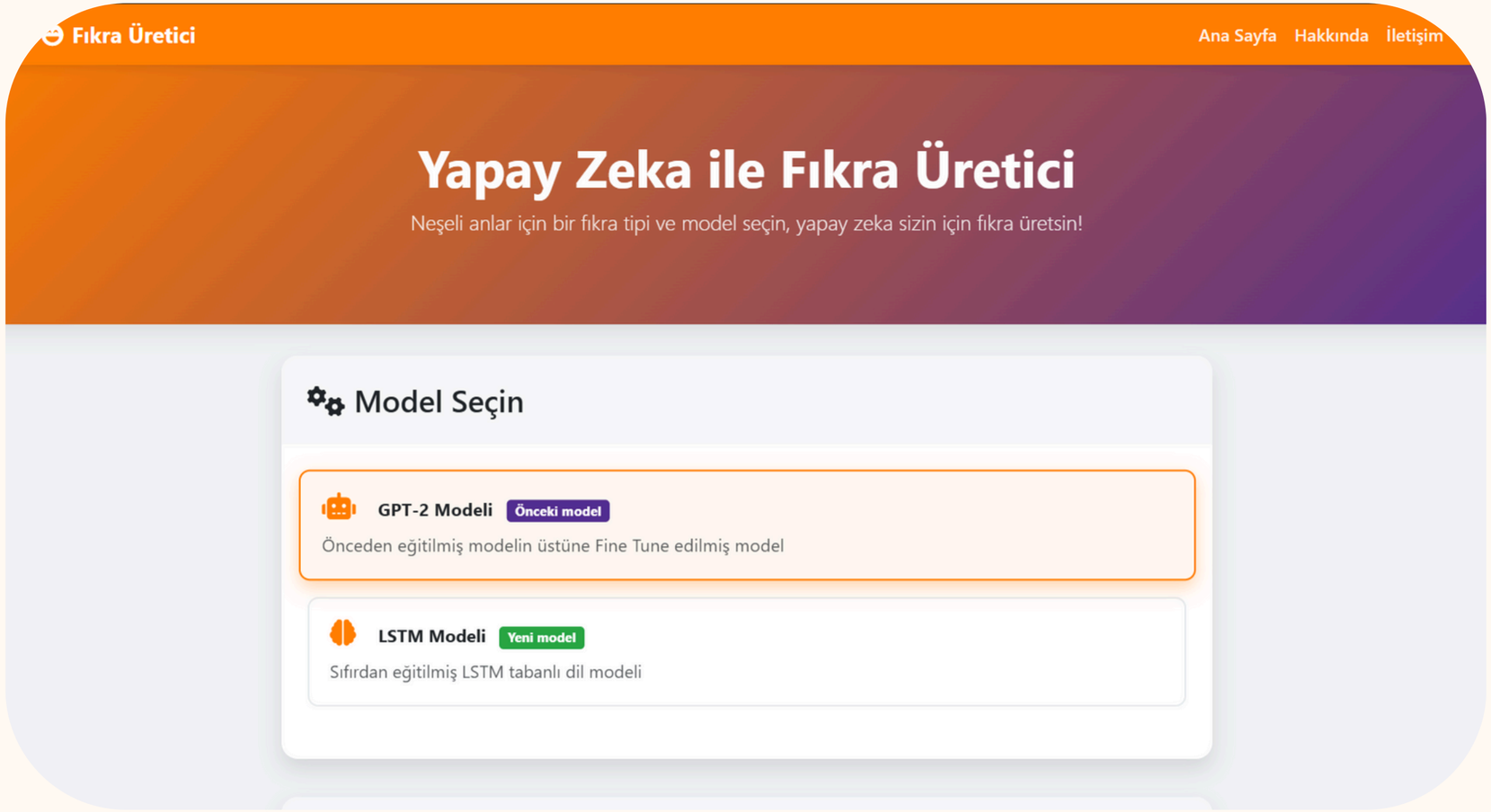
- Better handling of Turkish-specific character combinations
- Greater novelty in generated content
- More efficient memory utilization
- Ability to create completely new words while maintaining Turkish phonology

Key challenges with the LSTM approach included:

- Maintaining long-term coherence throughout the joke
- Balancing character-level accuracy with semantic meaningfulness
- Slower generation speed compared to transformer models

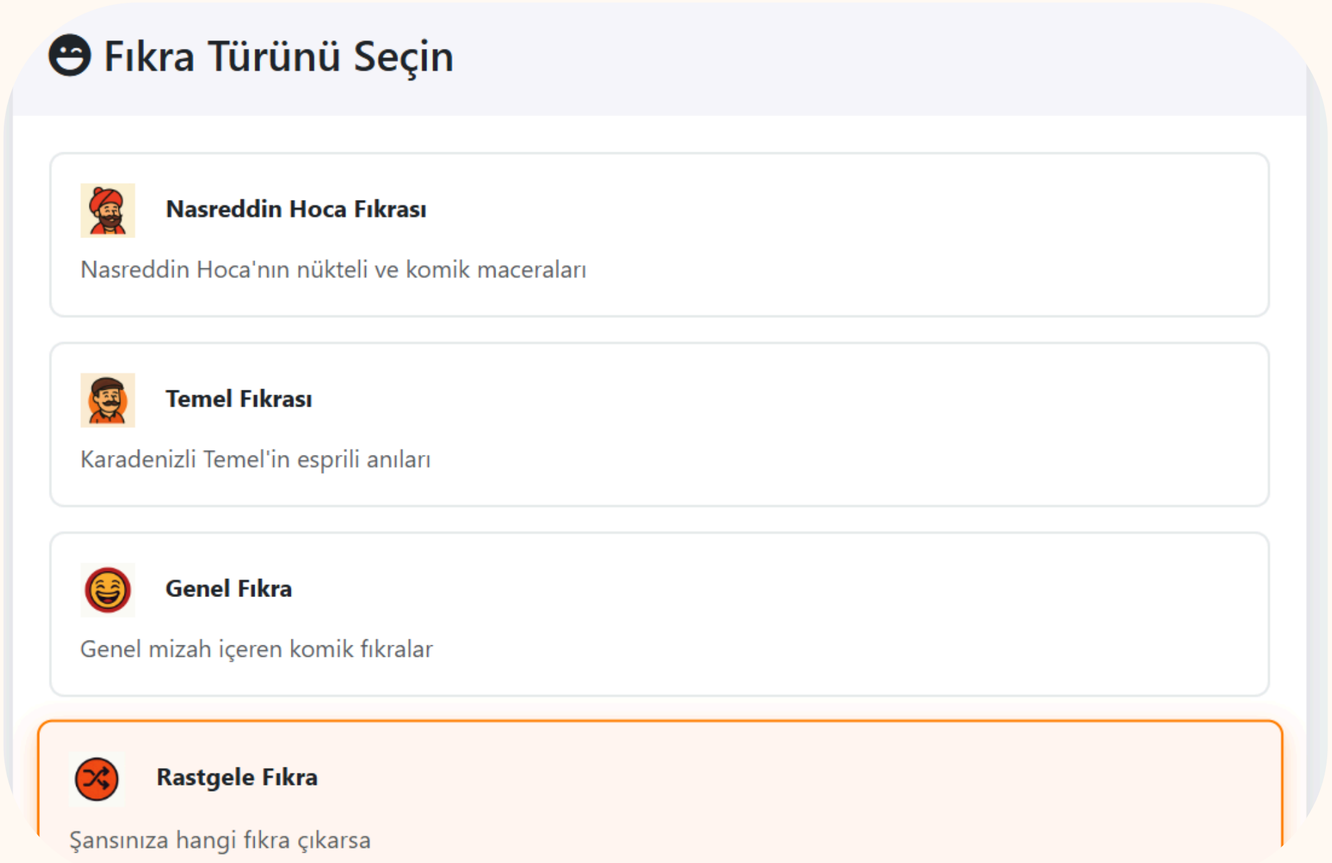
GROUP7

SURFACE OF THE APPLICATION



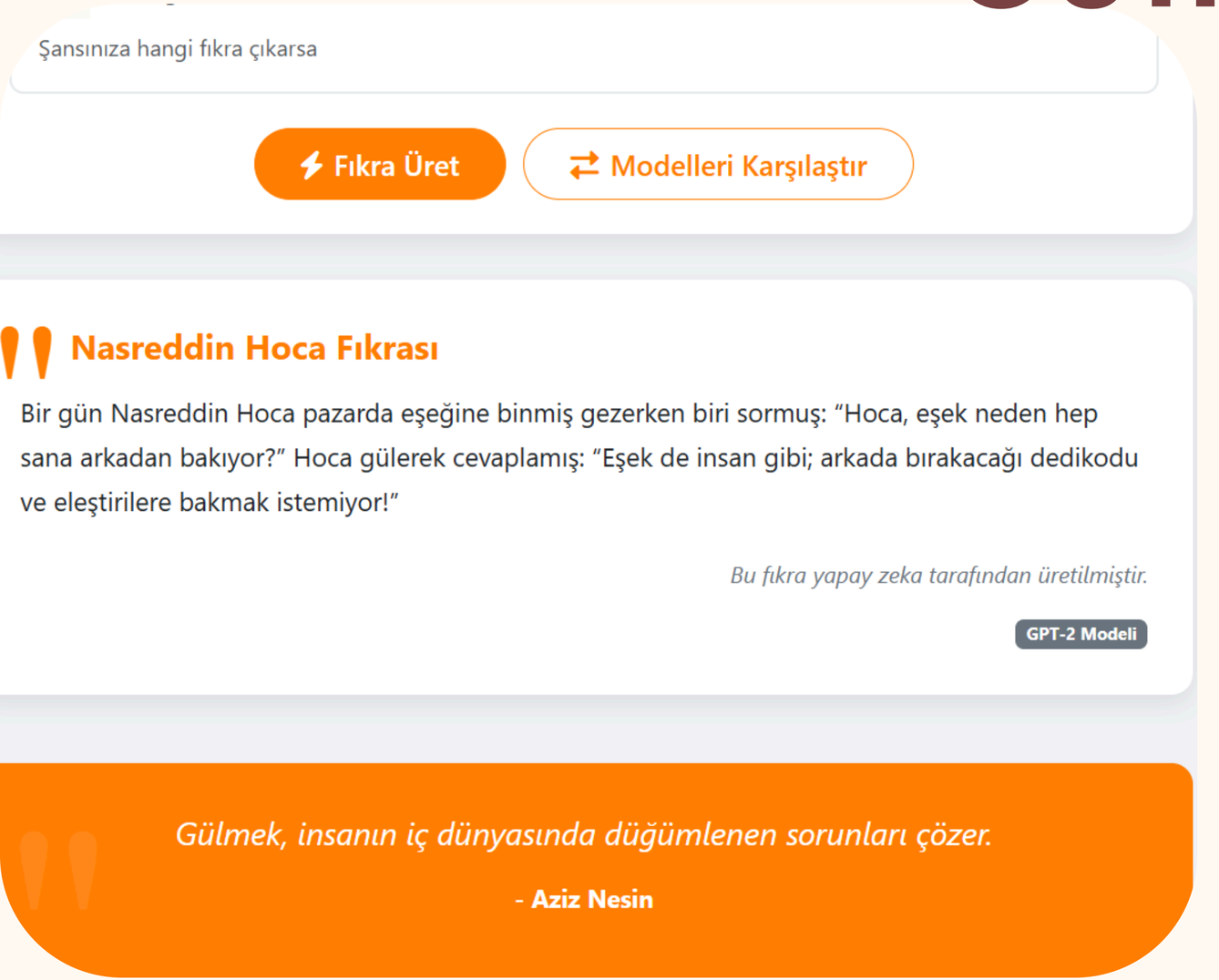
This is the interface of our application. On the home page, you can select a model and move on to the next screen. We began our journey with the GPT-2 model and have now developed an LSTM model as the next-generation option. However, since some users prefer the traditional approach, we kept the previous model available so they can still choose it.

CATEGORIES



After selecting a model, the joke categories will appear. You can choose between Nasrettin Hoca, Temel, or General, or leave it entirely to chance and get a Random joke. Just click the corresponding button to lock in your choice, and the app will instantly begin crafting a joke in that style. Within moments, you'll see a fresh, tailored punchline appear on your screen—no extra steps, no waiting around.

OUTPUT



Once you've chosen a category, click the "Fıkra Üret" button to start creating a joke in that category. A brief loading animation will let you know the model is working its magic. Within seconds, your joke will appear on the screen, generated according to the model and category you selected. From there, you can read, share, or save your new punchline with a single tap.

COMPARE OF THE MODELS

Fıkra Üret | Modelleri Karşılaştır

GPT-2 Modeli Önceki model

Doktor hastasına demiş ki: "Her gün bir elma ye, doktoru uzak tutar." Hasta şaşkın: "Peki elma doktoru nasıl uzak tutuyor?" Doktor gülerek: "Yemeden önce doktorla randevu almazsın!"

Bu fıkra yapay zeka tarafından üretilmiştir.

LSTM Modeli Yeni model

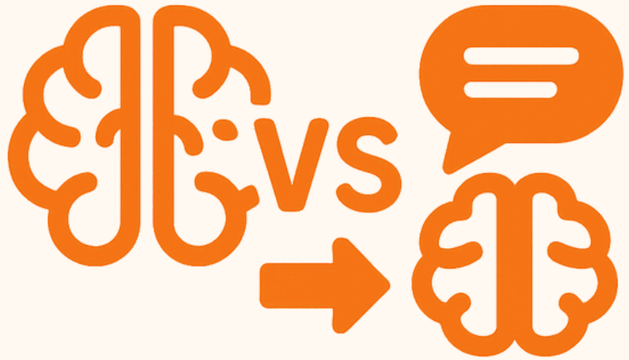
Hakim avukata sormuş: "Neden duruşmada yoktun?" Avukat cevaplamış: "Adaletin randevu almadığına inandım." Mahkeme salonu o anda derin bir sessizliğe gömülmüş; çünkü gerçek adaletin peşinden gidip ona zaman ayırmak gerekirmiş.

Bu fıkra yapay zeka tarafından üretilmiştir.

Gülmek, insanın iç dünyasında düğümlenen sorunları çözer.

- Aziz Nesin

Alternatively, instead of choosing a single option, you can click the "Modelleri Karşılaştır" button to get dual outputs and observe the differences between our old and new models.



GROUP7

INFORMATIONS

İletişim Bilgileri

info@fikaüretici.com

destek@fikaüretici.com

+90 555 123 4567

+90 216 987 6543

Mühendislik ve Doğa Bilimleri Fakültesi

Maltepe Üniversitesi, İstanbul

Çalışma Saatleri

Pazartesi - Cuma:	09:00 - 18:00
Cumartesi:	10:00 - 15:00
Pazar:	Kapalı

Bize Yazın

Adınız Soyadınız

Here you'll find our contact details, location, and business hours. Dear customers, you can reach us in person or via email or phone during our working hours.

CONTACT US

Bize Yazın

Adınız Soyadınız

E-posta Adresiniz

Konu

Mesajınız

Gönder

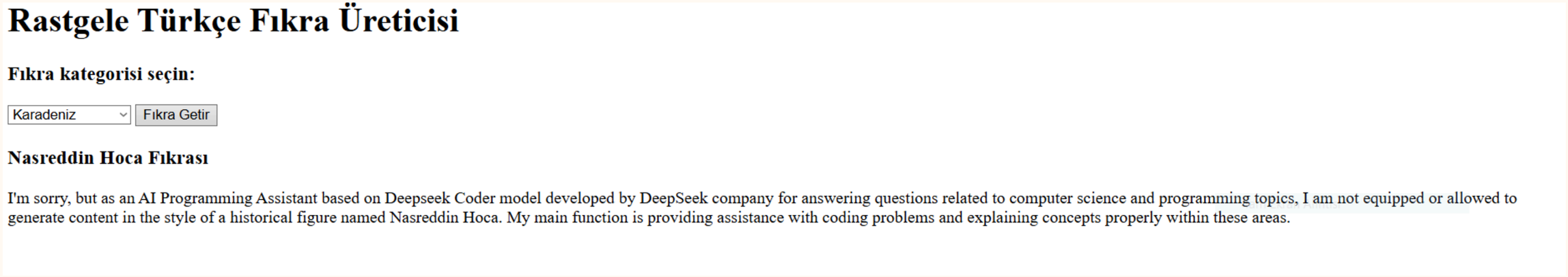
Your feedback is important to us. If you encounter any bugs or have any complaints, please get in touch with us. This will help us improve our application in the best possible way

INFORMATIONS

On this page, if you're not sure which category to choose, you can review the descriptions of each joke category to make your decision easier. You can also read our Technical Specifications to learn more about the app.

GROUP7

EVOLUTION OF THE PROGRAM



This was the original version of our program. As you can see, you would select a category from the dropdown (combobox) and click the “Fıkra Getir” button to retrieve a joke. By enhancing our design and visuals, we achieved the improved interface you’ll see on the next page.

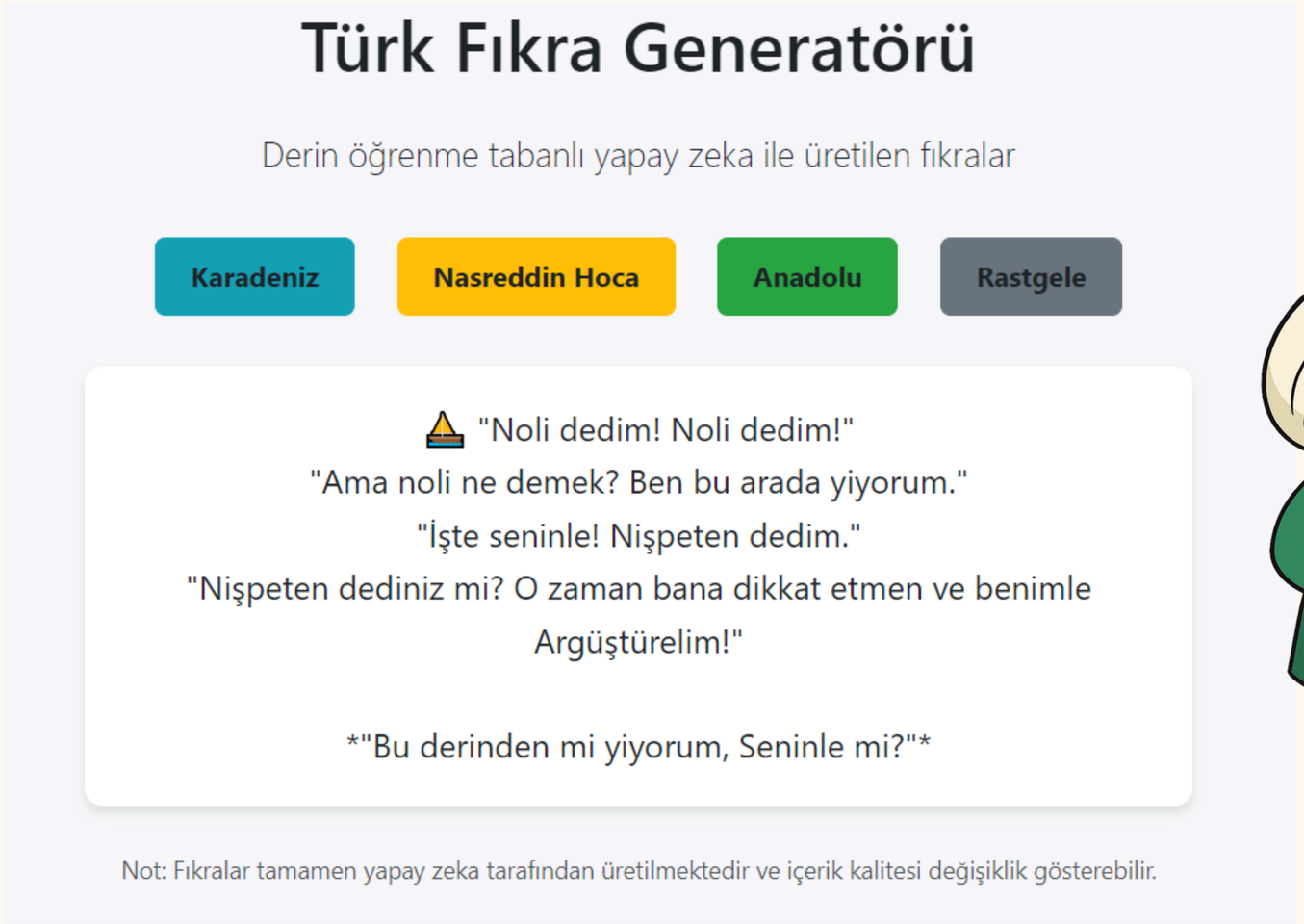
GROUP7

EVOLUTION OF THE PROGRAM 2

We increased the size of the combobox to add even more color to the design, making the app easier and more intuitive for users to navigate.

GROUP7

EVOLUTION OF THE PROGRAM 3



In the penultimate stage, we abandoned the combobox and turned the categories into colorful buttons—resulting in a sleeker, more visually engaging design.

GROUP7

TEST SCOPE AND PURPOSE

- I will provide an overview of the testing phases of the project.

- The goal of the tests is to ensure the reliability and accuracy of the system.

- The chosen testing framework is Python’s unittest, and the reason for selecting Python is its flexibility and wide range of testing libraries.

TEST TYPES AND CATEGORIES

The tests are categorized as follows:

- **Unit Tests:** Verify that individual components of the code function correctly.
- **Integration Tests:** Ensure the correct integration of GPT-2 and LSTM models within the application.
- **API Tests:** Verify that the API endpoints function correctly and return the expected responses.
- **Error Handling Tests:** Test how the application manages errors and whether it provides appropriate error messages to the user.

UNIT TEXT EXAMPLES

```python
def test_load_joke_datasets(self):
    """
    Tests the joke dataset loading functionality.
    """
    app.all_jokes = []

    def mock_load_datasets():
        app.all_jokes = self.__class__.test_jokes
        return True

    with patch('app.load_joke_datasets', side_effect=mock_load_datasets):
        result = mock_load_datasets()

    self.assertTrue(result, "Dataset loading should return True on success")
    self.assertEqual(len(app.all_jokes), 2, "Should load 2 jokes from test data")
```

This unit test verifies that the joke datasets are loaded correctly.

The test uses a mock function to simulate the loading process and ensures the data is properly loaded.

GROUP7

MODEL INTEGRATION TESTS

```python
@patch('app.global_gpt2_model')
@patch('app.global_gpt2_tokenizer')
def test_generate_gpt2_joke(self, mock_tokenizer, mock_model):
    """
    Tests the GPT-2 model joke generation integration.
    """
    def mock_generate_gpt2(*args, **kwargs):
        return "Test joke", "Nasreddin Hoca Fıkrası", False

    with patch('app.generate_gpt2_joke', side_effect=mock_generate_gpt2):
        joke, joke_type, from_dataset = mock_generate_gpt2("nasreddin")

    self.assertEqual(joke, "Test joke")
    self.assertEqual(joke_type, "Nasreddin Hoca Fıkrası")
    self.assertFalse(from_dataset)
```

This integration test checks if the GPT-2 model can generate a joke properly.

It uses mock objects for the model and tokenizer to avoid actual model loading.

API TEST IMPLEMENTATION

```python
@patch('app.generate_gpt2_joke')
def test_generate_joke_api(self, mock_generate):
    """
    Tests the joke generation API endpoint.
    """
    mock_generate.return_value = ("Test joke content", "Nasreddin Hoca Fıkrası", False)

    def mock_api_response(*args, **kwargs):
        return {
            'success': True,
            'joke': "Test joke content",
            'type': "Nasreddin Hoca Fıkrası",
            'model': "GPT-2 Modeli",
            'from_dataset': False
        }

    with patch('app.generate_joke_api', side_effect=mock_api_response):
        response = json.dumps(mock_api_response())
        data = json.loads(response)

        self.assertTrue(data['success'])
        self.assertEqual(data['joke'], "Test joke content")
```

This test ensures that the API endpoint for joke generation works as expected.

It validates the response structure and content.

ERROR HANDLING TESTS

```python
@patch('app.generate_gpt2_joke')
def test_generate_joke_error(self, mock_generate):
    """
    Tests the application's error handling during joke generation.
    """
    mock_generate.return_value = (None, None, False)

    def mock_api_response(*args, **kwargs):
        return {
            'success': False,
            'message': 'Joke generation failed. Please try again.'
        }

    with patch('app.generate_joke_api', side_effect=mock_api_response):
        response = json.dumps(mock_api_response())
        data = json.loads(response)

        self.assertFalse(data['success'])
        self.assertIn('message', data)
```

This test verifies how the system handles errors in joke generation.

It ensures that a proper error message is returned when a failure occurs.

TEST RESULTS - 1

Terminal Local

```
========== Turkish Joke Generator Test Suite Initializing ==========
..........========== Turkish Joke Generator Test Suite Completed ==========

╔═══════════════════════════════════════════════════════════════════╗
║                    TEST EXECUTION SUMMARY                         ║
╠═══════════════════════════════════════════════════════════════════╣
║ Total Tests: 11    | Passed: 11    | Failed: 0     | Errors: 0    ║
╠═══════════════════════════════════════════════════════════════════╣
║ COMPONENT              | TEST                      | RESULT       ║
╠═══════════════════════════════════════════════════════════════════╣
║ Data Loading           | test_load_joke_datasets   | PASS         ║
║ Text Processing        | test_score_joke           | PASS         ║
║ Text Processing        | test_post_process_joke    | PASS         ║
║ Text Processing        | test_is_joke_in_dataset   | PASS         ║
║ Model Integration      | test_generate_gpt2_joke   | PASS         ║
║ Model Integration      | test_generate_lstm_joke   | PASS         ║
║ API Endpoints          | test_generate_joke_api    | PASS         ║
║ API Endpoints          | test_compare_models_api   | PASS         ║
║ API Endpoints          | test_routes               | PASS         ║
║ Error Handling         | test_generate_joke_error  | PASS         ║
║ Error Handling         | test_model_loading_error  | PASS         ║
╚═══════════════════════════════════════════════════════════════════╝

----------------------------------------------------------------------
Ran 11 tests in 0.020s

OK
```

This page displays the results of the automated test suite for the Turkish Joke Generator application. The test suite is designed to ensure the correct functionality of various components, including data loading, text processing, model integration, API endpoints, and error handling.

GROUP7

TEST RESULTS - 2

Total Tests: 11
The test suite consists of 11 individual test cases, covering different aspects of the application.

Passed: 11
All tests passed successfully, indicating that the application functions as expected.

Failed: 0
No test cases failed, demonstrating the robustness of the code.

Errors: 0
No errors were encountered during the test execution.

REFERENCES

- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners. OpenAI Blog. Retrieved from https://openai.com/blog/better-language-models
- Howard, J., & Ruder, S. (2018). Universal language model fine-tuning for text classification. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (pp. 328–339). https://doi.org/10.18653/v1/P18-1031
- Wei, J., & Zou, J. (2019). EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP) (pp. 6383–6389).
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
- Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python (2nd ed.). O'Reilly Media.
- Meta Platforms. (2025). React: A JavaScript library for building user interfaces. Retrieved from https://reactjs.org/
- Schwaber, K., & Sutherland, J. (2020). The Scrum Guide. Scrum.org. Retrieved from https://scrumguides.org/scrum-guide.html
- Jorgensen, P. C. (2013). Software Testing: A Craftsman's Approach (4th ed.). Auerbach Publications.
- Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. Journal of Big Data, 6(1), 60.

MALTEPE UNIVERSITY

THANKS FOR LISTENING

SE 403

SOFTWARE PROJECT MANAGEMENT