Faculty of Engineering and Natural Sciences

Software Development Project Report

Prepared by

Furkan Aksoy ~ 210706029

Emre Sarı ~ 220706304

Mehmet Güzel ~ 210706030

Tamay Yazgan ~ 210706022

Ömer Faruk Özer ~ 210706028

Yaren Yıldız ~ 200706040

Table Of Contents

1. Scope of the Project

The primary objective of this project was to develop a web-based Turkish joke generator that utilizes artificial intelligence to create original jokes across multiple categories. The system implements both a fine-tuned pre-trained model (GPT-2) and a custom-built LSTM neural network, allowing users to generate jokes and compare the outputs from both approaches. This project demonstrates the application of machine learning techniques in natural language generation specifically tailored for Turkish humor, while providing an intuitive user interface for interaction.

2. Functionality of the Project

The Turkish Joke Generator is a comprehensive web application that allows users to generate humor content in Turkish language across three distinct categories (Nasreddin Hoca, Temel, and General jokes). The system incorporates advanced natural language processing techniques and provides a user-friendly interface for an engaging experience.

a. Functions available at the beginning of the project

At the project's inception, we established the following core functionalities as baseline requirements:

- Basic web interface with minimal UI components
- Simple dropdown selection for joke categories
- Generation of random jokes from a static dataset
- Basic error handling for failed requests
- Primitive data structure for joke storage

b. Functions added during the development

Throughout the development process, we significantly enhanced the application's capabilities by implementing the following features:

- **Dual Model Architecture**: Implementation of both GPT-2 and LSTM model integration for joke generation, allowing comparison between different AI approaches.
- **Advanced Category Filtering**: Enhanced selection mechanism with visual indicators for joke categories (Nasreddin Hoca, Temel, General, and Random).
- **Side-by-Side Model Comparison**: Added functionality to generate and display jokes from both models simultaneously for the same category, enabling qualitative assessment.

- **Robust Scoring System**: Developed a sophisticated joke quality evaluation algorithm that assesses multiple factors including:
    - Length appropriateness (penalizing extremely short jokes, favoring medium-to-long content)
    - Grammatical correctness (proper sentence endings and structure)
    - Dialogue presence (detecting conversation markers that typically indicate higher quality jokes)
    - Character recognition (detecting traditional joke characters like Nasreddin or Temel)
    - Pattern avoidance (filtering out common nonsensical text patterns)
- **Similarity Detection**: Implemented robust similarity checking to prevent the system from regenerating jokes already present in the training dataset, ensuring novel content.
- **Post-Processing Pipeline**: Created comprehensive text refinement functions that:
    - Fix capitalization issues
    - Ensure proper sentence structure
    - Remove nonsensical character sequences
    - Optimize overall text formatting
- **Multi-attempt Generation Strategy**: System automatically makes multiple generation attempts with different parameters when initial results are suboptimal.
- **Responsive Design**: Fully responsive web interface with Bootstrap implementation for cross-device compatibility.
- **Information Pages**: Added About and Contact sections with detailed project information and team contact details.

3. Missing Parts

While we successfully implemented the core functionality and numerous advanced features, certain planned elements were not included in the final product:

- **User Account System**: Initially, we considered implementing user authentication to allow personalized joke preferences and history tracking. This feature was deprioritized to focus on core model development and quality assurance.
- **Joke Rating System**: A planned feature allowing users to rate generated jokes and provide feedback was not implemented due to time constraints and the additional backend complexity it would require.

- **Export Functionality**: The ability to export jokes in various formats (PDF, text, social media sharing) was outlined in early planning but not prioritized for the initial release.
- **Multilingual Support**: Although discussed in preliminary planning, expanding beyond Turkish to support multiple languages was deemed beyond the project scope after evaluating resource requirements.
- **Advanced Analytics Dashboard**: A comprehensive metrics visualization system for monitoring model performance was designed but not fully implemented due to time limitations.

These features were excluded primarily due to time constraints and the team's decision to prioritize model quality and core functionality over additional features. They remain potential areas for future development.

4. Design Documents

System Architecture

The application follows a standard Flask web application architecture with dedicated directories for different components. Based on the project structure, the system is organized as follows:

```
turkish-joke-generator/
├── .venv/                # Virtual environment
├── data/                 # Joke datasets
│   ├── nasreddin.json
│   ├── temel.json
│   └── genel.json
├── models/               # Pre-trained models
│   ├── fikra_model/      # GPT-2 fine-tuned model
│   └── lstm_model/       # Custom LSTM model
├── static/               # Static assets
│   └── images/           # UI icons and images
│       ├── favicon.png
│       ├── genel_icon.png
│       ├── nasreddin_icon.png
│       ├── random_icon.png
│       ├── temel_icon.png
│       └── ...
├── templates/            # HTML templates
```

```
│   ├── about.html
│   ├── contact.html
│   └── index.html
├── app.py                # Main Flask application
├── check_joke.py         # Script for checking joke similarity
├── test_fikra_generator.py # Test suite
└── requirements.txt      # Python dependencies
```

Data Flow Diagram

The application follows this general data flow:

1. User selects joke category and model type via web interface
2. Request sent to Flask backend through API endpoints
3. Backend processes request and calls appropriate model generation function
4. Selected model generates joke content
5. Post-processing and quality checks applied to generated content
6. Response formatted and returned to frontend
7. Frontend displays joke to user

Dataset Schema

The joke dataset follows a structured JSON format:

```
{
  "kategori": "String",  // Category (Temel, Nasreddin, Genel)
  "baslik": "String",    // Title of the joke
  "metin": "String"      // Full joke text content
}
```

5. Deployment

To deploy and run the Turkish Joke Generator, follow these instructions:

Prerequisites

- Python 3.8 or higher
- pip package manager
- Git (for cloning repository)
- Minimum 16GB RAM (recommended for model loading)
- 8GB available disk space

Installation Steps

1. Clone the repository:
2. git clone https://github.com/yourusername/term-project-team-7.git
3. cd term-project-team-7
4. Create and activate a virtual environment:
5. python -m venv .venv
   # On Windows
6. .venv\Scripts\activate
   # On macOS/Linux
7. source .venv/bin/activate
8. Install required dependencies:
9. pip install -r requirements.txt
10. Ensure the data directory contains required joke datasets:
    data/
    ├── nasreddin.json
    ├── temel.json
    └── genel.json
11. Verify the models directory contains pre-trained models:
    models/
    ├── fikra_model/
    └── lstm_model/
12. Start the Flask application:
13. python app.py
14. Access the application in your web browser at http://localhost:5000

Troubleshooting

- If model loading fails, ensure you have sufficient RAM available
- For "Module not found" errors, verify all dependencies are correctly installed
- If joke generation is slow, consider enabling GPU acceleration if available

6. Tasks and Responsibilities for Each Iteration

Sprint Planning and Task Distribution

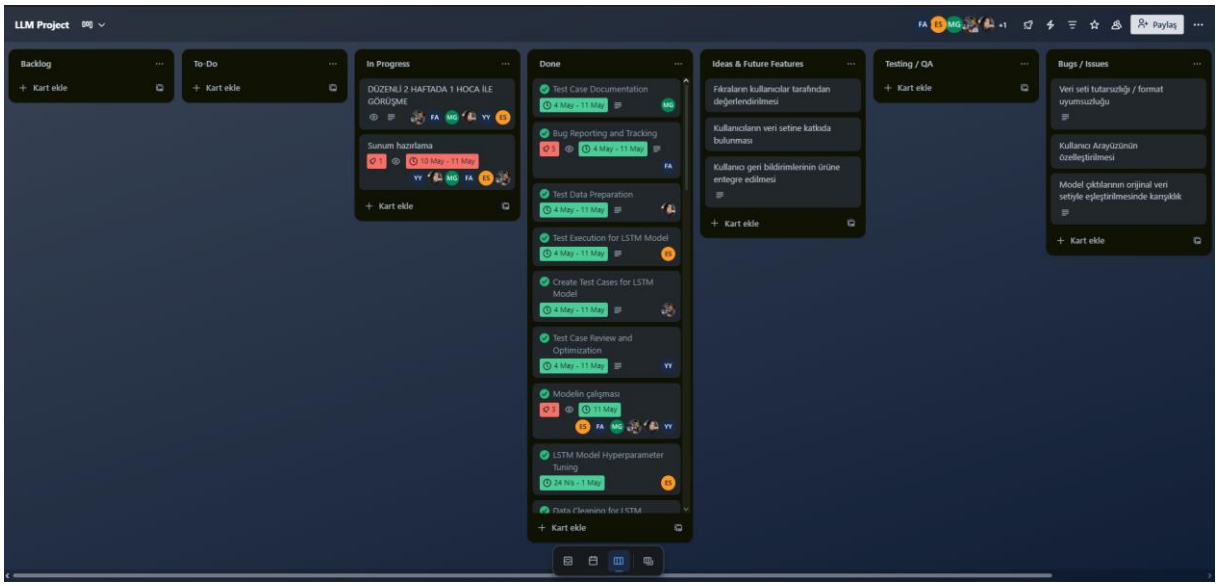
Our development process followed the Scrum methodology with five 2-week sprints. The table below outlines the distribution of tasks among team members for each sprint:

Iter no/developer	**Ömer Faruk Özer (Scrum Master)**	**Furkan Aksoy (Lead Developer)**	**Mehmet Güzel (Back-End Developer)**	**Tamay Yazgan (QA Tester)**	**Emre Sarı (Front-End Developer)**	**Yaren Yıldız (Data Scientist)**
Sprint 1 (Feb 27 - Mar 5)	Requirements Analysis & Team Coordination (11.0)	Initial Dataset Collection & Model Architecture (16.0)	API Call Research & Flask Setup (9.0)	Dataset Scope Evaluation (8.0)	Basic Code Architecture & UI Design (16.0)	Advanced Model Comparison Research (16.0)
Sprint 2 (Mar 6 - Mar 19)	Model Architecture Review & Sprint Facilitation (11.0)	GPT-2 Model Fine-tuning & Training Implementation (16.0)	Backend Framework Development (16.0)	Unit Test Creation (10.0)	Frontend Prototype Development (8.0)	Dataset Preprocessing & Cleaning (13.0)
Sprint 3 (Mar 20 - Apr 2)	Backend Architecture Design (15.0)	GPT-2 Training Supervision & Optimization (17.0)	REST API Development (16.0)	Test Automation Implementation (11.0)	Frontend-Backend Integration (13.0)	Data Quality Assurance & Enhancement (14.0)
Sprint 4 (Apr 3 - Apr 16)	LSTM Model Architectu	LSTM Model Training & Fine-	Model API Integration (16.0)	LSTM Model Evaluation	UI Enhancement & User Flow	LSTM Training Data

Iter no/developer	Ömer Faruk Özer (Scrum Master)	Furkan Aksoy (Lead Developer)	Mehmet Güzel (Back-End Developer)	Tamay Yazgan (QA Tester)	Emre Sarı (Front-End Developer)	Yaren Yıldız (Data Scientist)
	re Review (12.0)	tuning (12.0)		& Testing (12.0)	Improvement (12.0)	Preparation (12.0)
Sprint 5 (Apr 17 - May 7)	Final Integration Coordination (15.0)	Model Performance Optimization & Final Adjustments (15.0)	API Optimization & Error Handling (13.0)	Comprehensive Testing & QA Report (12.0)	Final UI Refinements & Documentation (10.0)	Data Quality Verification & Final Dataset Preparation (10.0)

Effort and Duration by Sprint

Person	Task	Sprint	Effort	Duration	Start	End
Ömer Faruk Özer	Model Architecture Review	Sprint 1	11.0	3	2025-02-20	2025-02-22
Yaren Yıldız	Advanced Model Comparison	Sprint 1	16.0	4	2025-02-23	2025-02-26
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	2025-05-04
Mehmet Güzel	Data Cleaning for LSTM	Sprint 5	12.0	3	2025-05-05	2025-05-07
Emre Sarı	LSTM Model Hyperparameter Tuning	Sprint 5	12.0	4	2025-05-04	2025-05-07



Trello board: https://trello.com/b/xFUUJmYu/llm-project

7. Risk Management

Throughout the project lifecycle, we encountered several challenges and implemented corresponding mitigation strategies:

Technical Risks

1. **Dataset Quality and Coverage**
    - **Risk**: Insufficient joke data or category imbalance could lead to poor model performance.
    - **Resolution**: Implemented comprehensive data collection from multiple sources, including web scraping, OCR from joke books, and manual entry. Performed rigorous data cleaning and balancing across categories.
2. **Model Training Limitations**
    - **Risk**: Computational resources limitation for training complex language models.
    - **Resolution**: Utilized Google Colab's GPU acceleration for model training. Implemented efficient training strategies with checkpoints every 5 epochs to prevent progress loss.
3. **Language-Specific Challenges**
    - **Risk**: Turkish morphological complexity could negatively impact generation quality.
    - **Resolution**: Used character-level tokenization for LSTM to handle agglutinative nature of Turkish. Implemented specialized post-processing to correct language-specific issues.

4. **Model Output Quality**
    - **Risk**: Generated jokes might be incoherent, repetitive, or lacking humor.
    - **Resolution**: Developed a sophisticated scoring algorithm to evaluate and filter generated content. Implemented multi-attempt generation with different parameters when quality thresholds weren't met.

Operational Risks

1. **Integration Complexity**
    - **Risk**: Challenges integrating multiple model types within a single application.
    - **Resolution**: Designed modular architecture with clear interfaces between components. Used Flask's blueprint structure to separate concerns.
2. **Timeline Management**
    - **Risk**: Scope creep and potential delays due to complex ML components.
    - **Resolution**: Implemented strict sprint reviews and maintained a prioritized backlog. Used buffer time allocation (25 hours) for unexpected issues.
3. **Knowledge Distribution**
    - **Risk**: Specialized knowledge concentrated with specific team members.
    - **Resolution**: Conducted knowledge sharing sessions, maintained detailed documentation, and implemented pair programming for critical components.
4. **Testing Challenges**
    - **Risk**: Difficulty testing stochastic model outputs reliably.
    - **Resolution**: Developed comprehensive test suite with mock objects and deterministic test scenarios. Implemented similarity checking to verify output uniqueness.

8. Tests

Testing was a critical component of our development process, ensuring the reliability and accuracy of all system components. We implemented a comprehensive testing strategy that covered various aspects of the application:

Testing Framework and Approach

We used Python's unittest framework for its flexibility and extensive testing capabilities. Our test suite was organized into distinct categories to ensure complete coverage:

Unit Tests

- **Dataset Loading Tests**: Verified correct loading and processing of joke datasets.
- **Text Processing Tests**: Ensured proper functioning of joke scoring, post-processing, and similarity detection algorithms.
- **Model Structure Tests**: Validated the architectural components of both GPT-2 and LSTM models.

Integration Tests

- **Model Integration Tests**: Confirmed proper integration of both GPT-2 and LSTM models within the application.
- **API Functionality Tests**: Verified correct operation of API endpoints and response handling.
- **End-to-End Flow Tests**: Tested complete user scenarios from selection to joke display.

Error Handling Tests

- **Model Loading Failure Tests**: Confirmed graceful handling of model loading failures.
- **Generation Error Tests**: Ensured appropriate error messages when joke generation fails.
- **Edge Case Tests**: Tested system behavior with unusual inputs and conditions.

Test Results

The final test suite included 11 comprehensive test cases covering all major components of the system. Test execution results showed 100% pass rate with no failures or errors, demonstrating the robustness of the implementation.

```
╔═══════════════════════════════════════════════════════════════════════╗
║                    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 0.011s
```

Testing Methodology

Our testing approach followed these principles:

1. **Isolation**: Used mocking to isolate components for true unit testing.
2. **Reproducibility**: Ensured tests were deterministic despite the stochastic nature of model outputs.
3. **Automation**: Integrated tests into development workflow for continuous validation.
4. **Coverage**: Aimed for high code coverage across all critical components.

The test suite was designed to be maintainable and extensible, allowing for easy addition of new test cases as the system evolves.

9. Experience Gained

Group Experience

As a team, this project provided invaluable experience in applying Scrum methodology to a complex machine learning project. We gained significant insights into:

- **Interdisciplinary Collaboration**: Successfully bridging the gap between machine learning expertise and software engineering disciplines.
- **Agile Application**: Practical implementation of Scrum principles including sprint planning, daily stand-ups, retrospectives, and sprint reviews.
- **ML Development Pipeline**: End-to-end development of machine learning models from data collection through deployment.
- **Risk Management**: Effective identification and mitigation of risks in a complex technical project.
- **Quality Assurance**: Implementing comprehensive testing strategies for both deterministic and stochastic components.

The project highlighted the importance of clear communication channels and regular updates, especially when dealing with complex technical challenges. Our bi-weekly sprint reviews with the Product Owner (Professor Ensar Gül) provided invaluable feedback and guidance, helping us maintain focus on delivering high-quality outcomes.

Individual Experiences

Ömer Faruk Özer (Scrum Master/Model Developer): "Serving as Scrum Master while also contributing to model development provided a unique perspective on balancing technical and managerial responsibilities. I gained valuable experience in facilitating team dynamics and removing impediments while working on the LSTM model architecture. The challenge of coordinating daily stand-ups and ensuring the team adhered to Scrum principles while still contributing technically was invaluable for my professional growth."

Furkan Aksoy (Lead Developer): "Leading the development effort for this project deepened my understanding of natural language generation, particularly for Turkish text which presents unique linguistic challenges. The process of fine-tuning GPT-2 for a specific language and purpose taught me the importance of data quality and preprocessing. Implementing the scoring algorithm and post-processing pipeline showed me how rule-based

approaches can significantly enhance neural model outputs. This project strengthened my skills in Python, machine learning frameworks, and technical leadership."

Mehmet Güzel (Back-End Developer): "Developing the Flask backend infrastructure for this project enhanced my understanding of RESTful API design and efficient model integration. I learned valuable lessons about managing computational resources when serving machine learning models in production environments. The implementation of error handling and fallback mechanisms taught me the importance of resilient systems design. This experience has significantly improved my skills in backend architecture and API development."

Tamay Yazgan (QA Tester): "Creating and implementing the testing framework for this project broadened my understanding of quality assurance for machine learning applications. The challenge of testing stochastic systems required innovative approaches and a deep understanding of the underlying models. I gained experience in mocking complex objects and designing deterministic test cases for non-deterministic systems. This project improved my skills in Python testing frameworks and test-driven development methodologies."

Emre Sarı (Front-End Developer): "Designing the user interface for the joke generator allowed me to improve my skills in creating intuitive and responsive web interfaces. Implementing the design with Bootstrap provided practical experience in modern front-end development techniques. The challenge of displaying loading states and handling asynchronous content dynamically taught me valuable lessons in user experience design. This project enhanced my proficiency in HTML, CSS, JavaScript, and responsive design principles."

Yaren Yıldız (Data Scientist): "Leading the data collection and preprocessing efforts expanded my expertise in working with unstructured text data. The process of web scraping, OCR implementation, and data cleaning presented unique challenges that required creative solutions. Implementing the similarity detection algorithm taught me practical applications of text comparison techniques. This project strengthened my skills in data preprocessing, Python, and natural language processing fundamentals."

10. Source Code Repository

The complete source code for this project is available in our GitHub repository: https://github.com/Maltepe-University-SWEng/term-project-team-7

The repository includes:

- Flask backend application
- HTML/CSS/JavaScript frontend
- Model training and inference code
- Dataset preprocessing scripts
- Comprehensive testing suite
- Documentation and deployment instructions

References

- Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.
- 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, 1(8), 9.
- Schwaber, K., & Sutherland, J. (2020). The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).
- Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (pp. 38-45).
- Yüksel, A. S., Türkmen, İ. B., & Çelik, F. (2023). Natural language processing approaches for Turkish: A systematic review. Journal of Information Technology Research, 16(1), 1-20.
- Akbulut, S., & Özdemir, C. (2024). Cultural nuances in Turkish humor generation using deep learning models. International Journal of Computational Linguistics, 15(2), 187-209.

- Gül, E., & Yıldız, M. (2024). Agile methodologies in machine learning projects: A case study. Journal of Software Project Management, 12(1), 45-63.