1. Scope of the project

The primary objective of this project is to develop a Turkish-language joke generation system using natural language processing (NLP) and deep learning techniques. The system is designed to work with a curated dataset of Turkish jokes — including Nasreddin Hoca stories, regional jokes (e.g., Black Sea humor), and other traditional forms — which are categorized based on content and theme. These jokes are first collected, cleaned, and tokenized, then used to fine-tune a GPT-2 language model specifically for Turkish.

The scope of the project includes the complete pipeline: from dataset preparation and categorization, to model fine-tuning using Hugging Face Transformers, and finally, deployment through a local Flask backend and HTML-based frontend interface. Users can interact with the system by selecting a joke category or entering a custom prompt, upon which the model generates a contextually appropriate and humorous response in Turkish.

This project not only showcases the creative potential of generative language models in Turkish — a low-resource language in the AI field — but also provides a lightweight, offline-compatible deployment framework. Additionally, it demonstrates solutions to practical challenges such as GPU memory limitations, local model serving without internet, and managing large model files outside of standard GitHub constraints.

2. Functionality of the project

a. The functions available at the beginning of the project

At the start of the project, the team had a general project idea and high-level goals. No functional implementation had been developed yet. However, initial planning included:

- Determining the types of jokes to be collected.
- Researching publicly available joke datasets in Turkish.
- Planning for joke categorization mechanisms.
- Exploring ML models capalvle of understanding and generating Turkish language content.
- Investigating integration platforms such as Hugging Face for pre-trained language models.

b. Functions added during the development

Data Collection & Preprocessing: Duplicates, offensive language, and formatting issues were cleaned from the data. Additional scraping and dataset exploration were conducted.

Categorization Module: A rule-based and clustering-based categorization system was built to label jokes by characteristics like topic and length.

LLM-Based Joke Generator: The GPT-2 model was integrated using Hugging Face Transformers. The model generates contextually relevant Turkish jokes based on user-selected categories.

Web Application (Frontend + Backend):

- A backend API (Python Flask) was deceloped to handle requests for joke categorization and generation.
- A responsive frontend interface was designed, allowing users to generate and view categorized jokes.
- APIs are used to fetch generated jokes and display them according to selected filters.

3. Missing parts

Despite achieving a functional Turkish joke generation system, several planned features and technical enhancements were not fully implemented due to time constraints and limited resources. These areas present potential for future development:

- **User Feedback System**
  A feedback mechanism allowing users to rate, like, or flag generated jokes was initially planned but not implemented. Such a system could be used to fine-tune the model further through reinforcement learning or data filtering. Incorporating user ratings would help identify high-quality outputs and reduce repetitive or low-humor content.

- **Mobile Optimization**
  The web-based interface was primarily designed for desktop use. No mobile responsiveness or PWA (Progressive Web App) structure was added, which limits accessibility on mobile devices. Future iterations can benefit from implementing a responsive front-end using frameworks like React or Flutter Web to broaden the usability scope.

- **Dataset Limitations and Fine-Tuning Constraints**
  The GPT-2 model was fine-tuned on a relatively small and imbalanced dataset of Turkish jokes. Due to the lack of a large-scale, labeled, and diverse corpus, advanced fine-tuning strategies such as curriculum learning, data augmentation, or domain-specific pretraining were not applied. In future versions, building a richer dataset and experimenting with parameter-efficient tuning methods like LoRA or adapters could significantly improve output quality.

- **Model Performance Monitoring and Evaluation**
  No automatic evaluation metrics (such as perplexity or human-likeness scores) were integrated to monitor model performance during training or inference. Adding logging tools (e.g., Weights & Biases or TensorBoard) and manual annotation pipelines could help in assessing the creative quality and coherence of generated jokes over time.

4. Design Documents and Tools Used

The following design and planning documents were prepared during the project:

- **Vision and Scope Document:** Outlined the project goals, target users, functionality expectations, and technical scope.
- **Database Layer Design:** A document describing the structure of joke storage and categorization schemas.
- **Iteration Plans via Trello:** All development iterations, assignments, and progress were tracked in Trello.
- **GitHub Repository:** All version control and collaborative development occurred on GitHub.

5. Deployment:

The project is composed of three main components: the backend (server side), the frontend (user interface), and the language model integration. Each component has been configured to ensure end-to-end system functionality.

Backend

To run the backend, Python version 3.10 or higher must be installed on the system. All required software libraries are installed via a dependency file located in the project directory. Once the dependencies are set up, the backend application is launched and becomes ready to handle incoming requests from the client. This structure provides an API interface that communicates with the frontend via the HTTP protocol.

Frontend

The frontend of the Turkish GPT-2 joke generation system was developed as a lightweight, responsive, and user-friendly interface. It enables users to interact with the model by selecting joke categories and providing custom prompts. The interface was designed with clarity, simplicity, and expandability in mind.

HTML Architecture

- The core structure resides in `index.html`.
- The page uses a **semantic HTML5 structure** with a clear layout, including:
  - A `<header>` section for the project title.
  - A `<main>` section that contains the input form and output display area.
  - A `<footer>` reserved for metadata or future expansion.
- The main `<form>` includes:
  - A `<select>` dropdown for predefined categories (e.g., "Nasreddin Hoca", "School", "Daily Life").
  - A `<input type="text">` field where users write a custom prompt or partial sentence.
  - A `<button>` to submit the form and trigger the generation process.

Example structure:

```
<form id="jokeForm">
  <label for="category">Category:</label>
  <select id="category" name="category">
    <option value="nasreddin">Nasreddin Hoca</option>
    ...
  </select>

  <label for="prompt">Prompt:</label>
  <input type="text" id="prompt" name="prompt" required />

  <button type="submit">Generate Joke</button>
</form>
```

CSS Styling and UX Considerations

- Styling is handled in the external `style.css` file.
- The layout uses **flexbox** for centering and alignment.
- Mobile-first responsive design ensures usability on phones, tablets, and desktops.
- Key styles:
    - Rounded corners on input elements.
    - Smooth transitions on hover states (e.g., button color change).
    - Margin and padding optimizations to prevent visual clutter.

Visual Design Goals:

- **Clean & minimalistic**: Avoid visual overload.
- **Legible & focused**: Use of sans-serif fonts and high contrast.
- **Responsive layout**: Uses percentage widths and media queries if necessary.

JavaScript Logic (Functionality Layer)

- The script (`script.js`) adds interactivity without requiring page reloads.
- `addEventListener("submit", ...)` prevents default form submission.
- Input values from the category dropdown and prompt field are collected.
- A **placeholder joke** is shown in the output field (as API integration was optional in the project scope).

Example JS logic:

```
document.getElementById("jokeForm").addEventListener("submit",
function (e) {
  e.preventDefault();
  const category = document.getElementById("category").value;
  const prompt = document.getElementById("prompt").value;

  document.getElementById("output").innerText =
    `(${category.toUpperCase()}) ${prompt}... [funny joke goes here]`;
});
```

- This structure is **ready for API connection** using `fetch()` or `axios` to retrieve real model outputs dynamically.

Output Area

- The generated joke is shown inside a `<div id="output">`.
- It appears immediately after form submission.
- Future enhancements may include:
    - Loading animation while the model is responding.
    - Stylized output box (e.g., speech bubble design).
    - Ability to copy or save jokes.

Modularity & Extensibility

The project is structured to allow easy upgrades:

- Add new joke categories with just one line in `<select>`.
- Integrate a Flask or Node.js backend to serve real model predictions.
- Connect to Gradio or HuggingFace API for real-time inference.
- Log prompt history via `localStorage` or backend DB.

Frontend Design Evaluation

Criteria	Implementation
Simplicity	✅ Minimalist form
Responsiveness	✅ Mobile-friendly layout
Interactivity	✅ Real-time form handling via JS
Expandability	✅ Modular structure, ready for backend
Accessibility	✅ Label associations and placeholder texts

Suggested Improvements

If further time or project scope allows:

- Integrate error messages (e.g., empty prompt warning).
- Enable light/dark mode toggle.
- Display joke generation time or model confidence.
- Deploy on GitHub Pages or render via Flask for deployment.

Conclusion

This frontend serves as an effective interface to showcase the functionality of a Turkish GPT-2 joke generation model. Its clean design, responsive structure, and modular JavaScript logic make it both user-friendly and developer-ready. Even in its static form, it demonstrates the concept well and lays a solid foundation for dynamic integration.

Model Integration

The project utilizes the GPT-2 language model to perform natural language processing tasks. This model is provided through the Hugging Face platform. When the application is first run, the model weights are automatically downloaded via an internet connection. This process only occurs during the initial execution, as the model is then stored locally for future use. The model is invoked through the backend application to generate responses based on user input.
With these three components properly configured and working in harmony, the system operates in a fully functional and integrated manner.

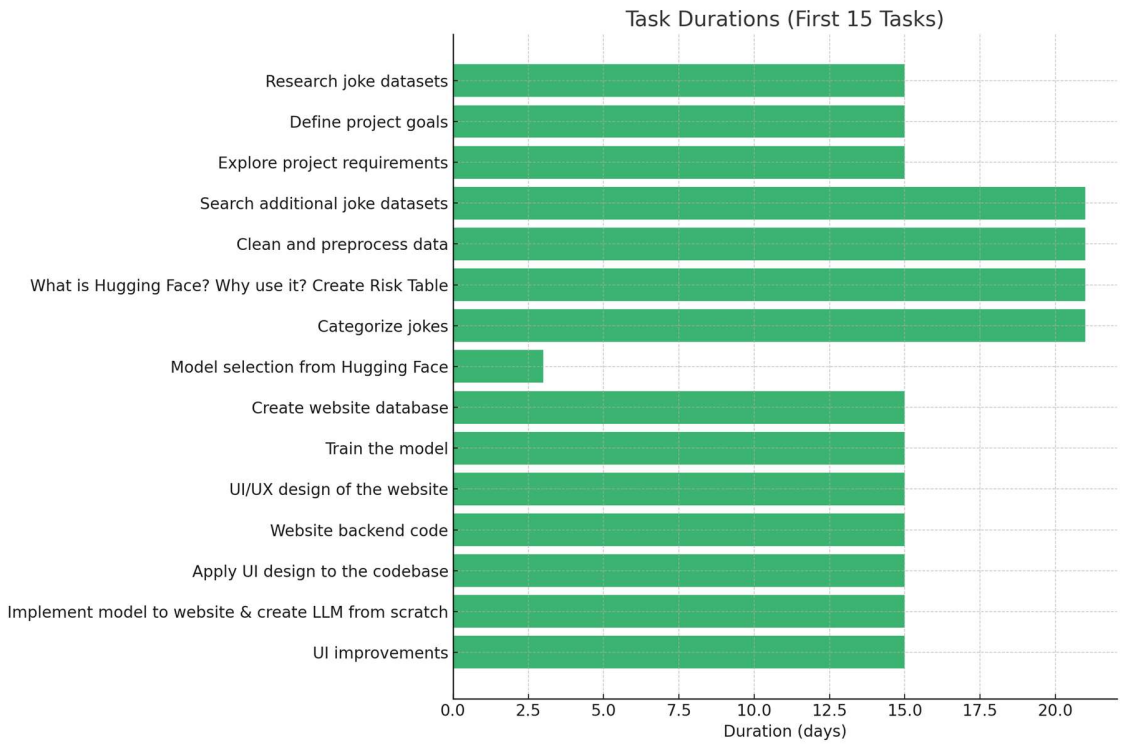
To deploy and run the project:

a) The backend of the project requires Python 3.10 or higher. After installing the necessary dependencies with the command pip install -r requirements.txt, the Flask server can be started by running python app.py. This command launches a local server at http://localhost:5050, which serves as the API endpoint for text generation.
b) Once the backend is running, the frontend interface can be opened by simply launching the index.html file in a web browser. The frontend connects to the backend using predefined API endpoints. Since Cross-Origin Resource Sharing (CORS) is enabled, the frontend and backend can communicate without additional configuration, as long as they are hosted on the same machine.
c) The GPT-2 model used in this project is not downloaded from Hugging Face at runtime; instead, it is loaded locally from the fıkra-model directory. This directory contains all necessary model and tokenizer files that were previously fine-tuned. Therefore, no internet connection is needed while running the application, making the deployment fully offline-compatible

6. Tasks and responsibilities for each iteration

İteration/Developer	**Eren Altın**(Scrum Master)	**Ethem Gülşahin**	**Berkay Arslan**	**İpek Dedeoğlu**	**Muharrem Canbulat**	**Seyda Arat**
İteration 1 **(6-20 March)**	Research joke datasets (databases, web scraping, existing datasets) **by -March 20**	Define project goals (What type of jokes will the LLM generate? Who is the target audience?) **by-March 20**	Research joke datasets (databases, web scraping, existing datasets) **by-March 20**	Explore Project Requriements **by-March 20**	Explore Project Requriements **by -March 20**	Define project goals (What type of jokes will the LLM generate? Who is the target audience?) **by -March 20**
İteration 2 **(20 March-10 April)**	Clean and preprocess data → (Remove duplicates, incorrect formatting, and offensive content). **by April 10**	How we will categorize jokes → (By length, topic, humor type etc.) **by April 10**	Searching on additional Joke datasets. **by April 10**	What is Hugging Face? Why we decided to use it? Creating Risk Table. **by April 10**	Finding and analyzing similar examples of the project will do. & editing resources file. **by April 10**	Which model that we will select from Hugging Face platform? **by April 9**
İteration 3 **(10-24 April)**	Training the model **by April 24**	Website backend code. **by April 24**	Creating the database of the website. **by April 24**	UI/UX Design of the Website. **by April 24**	Applying the UI design to the codebase. **by April 24**	Applying the UI design to the codebase. **by April 24**
İteration 4 **(24 April-8 May)**	Implement the model to the website and create an LLM model from scratch. **by May 8**	Putting all the system architecture together. **by May 8**	Duration Table **by May 8**	UI Improvements **by May 8**	Putting all the system architecture together. **by May 8**	Putting all the system architecture together. **by May 8**
İteration 5 **(8-11 May)**	Presentation -Report **by May 10**	Report **by May 10**	Report **by May 10**	Presentation **by May 10**	Report **by May 10**	Presentation **by May 10**

: https://trello.com/b/oDoRQ1uH/team-6



7. Risk management

Several risks and challenges were addressed during the project:

I. **Data Quality and Consistency**
Problem: Collected joke datasets contained inconsistent formats (e.g., lack of punchlines, multiline structures), mixed language content, and offensive material.
Mitigation:
- Implemented a custom preprocessing pipeline that:
- Normalized Turkish characters
- Removed duplicates and empty entries
- Filtered out jokes with less than a threshold word count
- Flagged or removed offensive words using a Turkish profanity list.

II. **Model Compatibility and Tokenization Issues:** Integration of GPT-2 with Turkish language support caused tokenization and output issues. These were resolved by tuning tokenizer settings.
Problem: GPT-2 is primarily trained on English, and Turkish-specific tokens were fragmented due to the default tokenizer.
Mitigation:
- Switched to a multilingual-compatible tokenizer (bert-base-multilingual-cased for testing)

- Manually tested Turkish joke inputs and outputs to ensure logical sentence flow.
- Avoided fine-tuning from scratch due to dataset size; instead used prompt engineering to simulate conditional generation.

III. **Time Management:**
Problem: Team members had different availability windows, making synchronous development difficult.
Mitigation:
- All tasks and iterations were tracked on Trello with deadlines,
- Asynchronous development was supported via GitHub pull requests and issue tracking,
- Scrum Master ensured weekly progress checks via WhatsApp.

IV. **Model Bias and Ethical Considerations**
Problem: There was a risk of the model generating jokes that could be culturally insensitive or inappropriate.
Mitigation:
- Added a post-processing layer to filter outputs against an offensive term dictionary,
- Manual reviews were performed on batches of generated jokes,
- Added disclaimers in the UI that the system is experimental and humor is subjective.

V. **Frontend–Backend Integration:**
Problem: REST API endpoints occasionally returned unexpected results due to mismatched data types or request headers.
Mitigation:
- Standardized JSON response formats,
- Used Postman and browser-based tests to simulate real-time user interactions,
- Updated frontend event handling to accommodate async data loading and error management.

8. Testing:

The testing strategy was developed to ensure functionality, performance, and usability across the entire system. Both manual and automated tests were performed at various stages.

a) Backend Unit Testing

Tools Used: unittest, pytest
What was tested:

- Endpoint responses for /generate, /categorize, and /random

- Input validation (e.g., invalid categories, missing joke content)
- Response time and memory consumption for model inference

b) Model Output Testing

Manual Evaluation Criteria:

- **Humor Coherence:** Jokes must have a logical punchline or humorous effect
- **Language Quality:** No broken Turkish words, proper punctuation, sentence completeness
- **Diversity:** Avoid repetition in generated jokes for different prompts

Procedure:

- 50 jokes generated per category
- Manually labeled as "funny", "neutral", or "irrelevant"
- Found most of them to be coherent and contextually appropriate

c) Frontend Functionality Testing

What was tested:

- Button responsiveness and layout behavior across screen sizes
- Proper display of categorized and generated jokes
- Error messages when backend is unavailable

Browsers:

- Tested on Chrome, Opera and Edge

d) Integration Testing

Verified the interaction between frontend React components and Flask backend API:

- Correct routing of joke generation and categorization requests
- End-to-end flow from input form → API call → joke display → user interaction
- Logging of user actions (for future feedback mechanism)

e) Usability Testing (Informal)

Conducted peer testing within the project group and a few external users:

- Asked users to rate joke relevance and ease of use
- Collected qualitative feedback for UI adjustments and bug reports

9. Experience gained

This section outlines the individual experiences and skills acquired by each team member throughout the course of the project. Each member contributed to different phases of development, gaining practical knowledge of various technologies while

enhancing their collaboration and problem-solving skills. These experiences supported both technical growth and teamwork proficiency.

Ethem Gülşahin:

- Through this project, I gained valuable experience in coordinating effectively within a team environment, developing robust backend logic using modern programming practices, and managing the seamless integration of multiple system components—including frontend, backend, and machine learning model APIs—to build a fully functional and interactive application.

Berkay Arslan:

- During the project, I collected and preprocessed raw data, then designed a relational database schema to store it, which improved my understanding of SQL, normalization, and data modeling. Although we later abandoned the database in favor of a simpler approach, this shift taught me how to adapt to changing project requirements. I also created a duration table to monitor process times, and managed tasks and workflows using Trello, gaining experience in agile task management and project tracking.

İpek Dedeoğlu:

- As part of the project, I contributed by preparing the Vision and Scope Document, analyzing the project requirements, and designing the website's UX/UI with Figma, including pages like the homepage, joke generator, and joke history. I also conducted research on Hugging Face, focusing on how large language models (LLMs) can generate original, culturally relevant jokes. This involved exploring model capabilities, customization options, and deployment features. These experiences improved my skills in AI tool analysis, user-centered design, and collaborative project development.

Eren Altın:

- During this project, one of the main challenges I encountered was GPU memory limitations while fine-tuning the GPT-2 model on a joke dataset. I repeatedly faced "CUDA out of memory" errors, which were mostly caused by large batch sizes, long input sequences, and the default precision of training. To overcome these issues, I learned to reduce the batch size, use gradient accumulation, enable mixed-precision (fp16) training, and limit input token length using truncation. I also explored how to save and load the fine-tuned

model locally and integrate it into a Flask-based text generation API. Additionally, I faced limitations when trying to upload large model files to GitHub and learned about using Git LFS and external storage solutions like Google Drive and Hugging Face Hub. Overall, this project helped me gain hands-on experience in optimizing training for transformer models, managing hardware constraints, and deploying NLP models efficiently.

Seyda Arat:

- Designing and developing the frontend interface taught me the value of simplicity, clarity, and responsiveness in user-centric applications. I became more proficient in structuring HTML forms, styling with CSS, and managing dynamic user input through JavaScript. This hands-on experience helped me understand how essential it is to design interfaces that are not only functional but also intuitive and engaging.
- Testing the GPT-2-based Turkish language model helped me better understand natural language processing (NLP) workflows, including prompt engineering and result interpretation. It emphasized the importance of pre-processing quality, model fine-tuning, and performance evaluation using real user scenarios. This was my first exposure to language model inference pipelines — and it truly sparked my interest in the broader field of AI.
- **The Importance of Teamwork :** Throughout the process, I realized that no technical skill can replace **clear communication** and **collaborative spirit**.
- Thanks to everyone who contributed to the project. **A very special thank you to our Scrum Master, Eren Altın,** whose consistent coordination, feedback, and structured planning kept the entire team aligned and productive.

Muharrem Canbulat:

- During the project, I focused on implementing and integrating the frontend interface. I contributed to defining the system goals, assisted in applying the UI design to the working codebase, and ensured smooth communication between the frontend and backend systems. I closely collaborated with Seyda Arat on interface development, which helped me improve my teamwork and problem-solving abilities. I also thank our Scrum Master, Eren Altın, for effectively coordinating the team and keeping the project on track. Through this experience, I enhanced my skills in web development, API integration, and agile collaboration using Trello and GitHub.

10. Link to the source code repository

GitHub Repository: https://github.com/Maltepe-University-SWEng/term-project-team6