**yazı tipi, metin, logo, grafik içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.**

**MALTEPE UNIVERSITY FACULTY OF NATURAL SCIENCES AND ENGINEERING**

**SE 403 Software Project Management Term Project**

**AI-Powered Joke Teller**

Instructor: Ensar Gül

Group Members:

|  |  |
| --- | --- |
| Melek Sude Günen | 220706052 |
| Yağmur Tank | 210706010 |
| Harun Yahya Ünal | 210706015 |
| Yasin Eren Şahin | 200706007 |
| Fadıl Ahmet Tüfekçi | 210706031 |
| Emir Adıgüzel | 210706011 |

Content

1.Scope of The Project2

1.1 Primary Objective2

1.2 Scope of The Project2

1.3.Out of Scope Features3

**2. Functionality of the AI-Powered Joke Teller Project3**

2.1. Initial Functionality3

2.2. Enhanced Functionality 4

**3. Missing Parts: Unimplemented Features5**

**4.Design Documents7**

4.1.System Architecture Diagram 7

4.2 Use Case Diagram8

4.3 Sequence Diagram9

**5.Deployment9**

[5.1.Required Software Installation 9](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775005)

[5.2.Aplication Setup 1](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775006)0

[5.3.Generate Joke 10](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775007)

[6. Tasks and Responsibilities 11](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775009)

[6.1.Effort and Duration 11](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775005)

[6.2.Task Breakdown 12](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775006)

[6.3.Effort and Duration Summary 13](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775007)

[6.4.Team Responsibilities by Iteration 13](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775007)

[7. Risk Management 14](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775010)

[8. Testing Methodology 17](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775011)

[9. Experience Gained 20](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775012)

[9.1.Group Experience 20](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775005)

[9.2.Individual Experiences 21](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775006)

[9.3.Challenges Faced 22](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775007)

[10. Conclusion 23](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775014)

[11. Source Code 24](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775014)

[12. References 24](file:///C:\Users\hamdi\Downloads\LLM_project_report.docx#_Toc197775014)

1.Scope of the Project

**1.1. Primary Objective**

The **AI-Powered Joke Teller** project aims to develop an interactive, AI-driven system that generates and delivers humorous content dynamically or retrieves jokes from a predefined database. The primary goal is to provide users with an engaging, personalized, and ethical joke-telling experience using Natural Language Processing (NLP) and Large Language Models (LLMs).

Key objectives:

* **Entertainment**: Deliver fun, AI-generated jokes to users through a seamless interface.
* **Personalization**: Allow users to Request new jokes with a single click.
* **Ethical AI**: Ensure inclusivity and appropriateness via content moderation and user reporting tools.
* **Scalability**: Support future enhancements like multilingual support and voice integration.

**1.2. Scope of the Project**

**In-Scope Features**

1. **Core Functionality**
   * **Joke Generation**:
     + Dynamic joke creation using open-source LLMs (e.g., YTU-CE-COSMOS GPT-2).
     + Retrieval from a categorized joke database.
   * **User Interaction**:
     + Button-triggered joke delivery with <2-second response time.
2. **Content Safety & Moderation**
   * AI-powered filtering of offensive/profane content.
   * Automated removal of flagged jokes and user reporting mechanisms.
3. **Technical Implementation**
   * Backend: Python (Flask)
   * Frontend: Responsive web UI (React/Next.js/JavaScript/HTML/ Tailwind CSS).
   * Deployment: Cloud-based or local hosting for scalability.
4. **Future-Readiness** (Phased Releases)
   * **Phase 1**: Text-based joke generation (MVP).
   * **Phase 2**: Advanced personalization.

**1.3.Out-of-Scope Features**

* Video/animated joke generation.
* Real-time emotional recognition or sarcasm detection.
* Advanced chatbot interactions beyond joke delivery.

The project’s scope is tightly focused on delivering a lightweight, entertaining, and ethically constrained AI joke-telling system. By leveraging open-source NLP tools (Hugging Face) and iterative development, it balances immediate functionality with long-term adaptability for diverse audiences.

**2. Functionality of the AI-Powered Joke Teller Project**

The AI-Powered Joke Teller is designed to deliver humor through an interactive, AI-driven interface. Below is an overview of its core features and how they evolved during development.

**2.1. Initial Functionality**

When development began, the system included the following foundational features:

**a.Joke Generation & Retrieval**

* + **Dynamic Joke Creation**: The system used a pre-trained language model to generate jokes from scratch based on user prompts.
  + **Predefined Joke Database**: A collection of categorized jokes (e.g., Nasrettin Hoca, Soldier,Police jokes) was available for quick retrieval.

**b.Basic User Interaction**

* + **"Tell Me a Joke" Button**: A simple interface allowed users to request jokes with a single click.
  + **Text Display**: Generated jokes were shown in a clean, readable format.

**c.Technical Backbone**

* + **Backend API**: A lightweight server (Python/Node.js) handled joke requests.
  + **Model Integration**: The system connected to an open-source LLM (e.g., Hugging Face) for text generation.

**2.2. Enhanced Functionality (Added During Development)**

As the project progressed, the following improvements and new features were introduced:

**a.Personalized Humor**

* + **Joke Preferences**: Users could select favorite categories (e.g., "Puns Only") to tailor joke suggestions.
  + **Adaptive Feedback**: The system learned from user ratings (like/dislike) to refine future recommendations.

**b.Smarter Content Safety**

* + **AI Moderation**: Advanced NLP techniques were added to detect inappropriate context .

**c.Performance Upgrades**

* + **Response Time Optimization**: Model fine-tuning and caching reduced latency to under 2 seconds per joke.
  + **Error Handling**: The system provided friendly messages if joke generation failed .

**d.User Experience Refinements**

* + **Multilingual Testing**: Support for non-English characters (e.g., Turkish "ç, ı, ş") was validated.
  + **Repetition Control**: The system avoided reusing the same punchline excessively in a single session.

**e.Future-Ready Extensions**

* + **Multilingual Jokes**: Initial datasets for Spanish and Turkish jokes were prepared for future updates.

**Key Innovations**

* **Hybrid Approach**: Combines AI creativity with human-curated jokes for reliability.
* **Ethical Safeguards**: Proactive filtering and user reporting ensure inclusive humor.
* **Scalable Design**: Modular architecture supports easy addition of features like voice or new languages

**3. Missing Parts: Unimplemented Features**

Despite the project’s successful implementation, several planned features were not included in the final release. Below are the key omissions and the reasons behind their exclusion:

**a. Advanced Sarcasm & Tone Detection**

* **Planned Feature**: The system was intended to recognize sarcasm or adjust joke tone based on user interaction.
* **Reason for Exclusion**:
  + Limited NLP capabilities in open-source models made accurate sarcasm detection unreliable.
  + Prioritized core joke generation over complex tone analysis to meet deadlines.

**b. Real-Time Emotional Recognition**

* **Planned Feature**: Using camera or voice input to detect user emotions and tailor jokes accordingly.
* **Reason for Exclusion**:
  + Hardware dependencies (e.g., webcam/microphone access) added complexity.
  + Privacy concerns and ethical considerations around emotion tracking.

**c. Video/Animated Joke Delivery**

* **Planned Feature**: Jokes accompanied by animations or short video clips for enhanced engagement.
* **Reason for Exclusion**:
  + High development effort for multimedia content creation.
  + Focus remained on text-based humor to ensure lightweight performance.

**d. Deep Customization (User-Created Joke Styles)**

* **Planned Feature**: Allowing users to train the AI with their own humor preferences or writing styles.
* **Reason for Exclusion**:
  + Risk of generating inappropriate content from unvetted user inputs.
  + Computational costs of personalized model fine-tuning were prohibitive.

**e. Standalone Mobile App**

* **Planned Feature**: A dedicated iOS/Android app for on-the-go joke access.
* **Reason for Exclusion**:
  + Web-based deployment was prioritized for broader accessibility.
  + Resource constraints limited cross-platform development.

**f. Community Joke Submission Hub**

* **Planned Feature**: A platform for users to submit and vote on crowd-sourced jokes.
* **Reason for Exclusion**:
  + Moderation challenges for user-generated content.
  + Potential legal risks around unvetted joke ownership.

**Key Reasons for Omissions**

* **Technical Limitations**: Open-source models lacked specialized capabilities (e.g., sarcasm detection).
* **Ethical/Risk Factors**: Features like emotion tracking raised privacy concerns.
* **Resource Prioritization**: Core functionality (text jokes, moderation) took precedence over experimental additions.
* **Time Constraints**: Phased development focused on the MVP first.

**4.Design Documents**

**metin, ekran görüntüsü, diyagram, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.4.1.** **System Architecture Diagram**

The diagram illustrates a user interface system that processes HTTP requests to generate plain text responses. The workflow begins when a user submits a request through the interface. This request is then sent to a backend API, which performs two main actions: reading data and generating output. The system retrieves information based on an ID or category, likely querying a database or stored files. In this case, the data source is a collection of joke text files (TXT). The backend uses the "YTU Turkish-GPT2" model, a language model trained for Turkish text generation, to produce the final text output. The result is returned as a plain text response to the user, completing the cycle. This setup suggests a streamlined application, possibly a joke generator or text-based service, leveraging AI for content creation in Turkish.

**4.2 Use Case Diagram**

**metin, yazı tipi, ekran görüntüsü, beyaz içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.**

Use Case Description: Joke Generator

1. Actor

* User: The only human actor who interacts with the system by pressing a button to request a joke.

2. Core Use Case

* "Generate Joke":
  + Trigger: User clicks a button (e.g., "Tell me a joke").
  + Process: The system dynamically generates a joke using the YTU Turkish-GPT2 model.
  + Output: Returns a plain text joke (no filters, ratings, or moderation).

3. Technical Implementation

* Single API Endpoint:
  + Backend receives the request and forwards it to the AI model.
  + No authentication or complex routing needed (minimalist design).
* Model:
  + YTU Turkish-GPT2: Fine-tuned for Turkish joke generation.
* Response:
  + Plain text format.

4. Key Simplicity

* No extra steps (e.g., filtering, user feedback, or categories).
* Aligns with your "just generate a joke" goal.

**4.3 Sequence Diagram**

**metin, ekran görüntüsü, diyagram, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.**

The sequence diagram illustrates the end-to-end workflow of the AI-powered joke generation system. The process begins when a user clicks the "Generate Joke" button on the frontend interface. This action triggers an HTTP GET request to the /generate-joke endpoint in the Backend API. The backend then communicates with the YTU Turkish-GPT2 model, which processes the request and returns a generated joke in plain text format. The system incorporates an optimization feature where it first checks for cached jokes in the TXT files database before generating a new one. If a suitable cached joke exists, it's retrieved from storage; otherwise, a fresh joke is generated by the AI model. Once obtained, the joke is sent back as a raw text response to the frontend, which finally displays it to the user. The technical implementation relies specifically on the YTU Turkish-GPT2 model for joke generation, ensuring culturally relevant Turkish humor output. The entire process emphasizes simplicity and efficiency, with the optional caching mechanism providing potential performance benefits for frequently requested jokes while maintaining the core functionality of instant AI-generated humor delivery.

**5.Deployment**

**5.1.Install Required Software:**

* Python 3.8 or newer (download from https://www.python.org/downloads/)
* Git (to download the project)
* 7GB+ free space (for model files)
* Node.js (LTS version, Download from <https://nodejs.org/tr>)

**Recommended:**

* NVIDIA GPU with CUDA support (for faster performance)
* Linux/macOS (Windows may need extra setup)

**5.2.Application Setup**

* Clone the repository
* Setup backend
* Create and activate virtual environment (Recommended)

Windows: .\venv\Scripts\activate

macOS/Linux: source venv/bin/activate

* Run flask server

Runs at: <http://localhost:5000>

* Test API: <http://localhost:8080/joke>
* Run backend by executing JokeServer.py
* Frontend setup
* Node.js dependencies (install npm)
* Navigate to the frontend directory: cd frontend
* Install Node.js dependencies: npm install
* Start the React development server: npm run dev
* Application will be available at: http://localhost:3000
* The frontend will automatically connect to the backend API running on port 5000

**Note:**

* Ensure both backend (Flask) and frontend (React) servers are running simultaneously
* The React development server will automatically reload when you make changes to the frontend code
* For production deployment, you would need to build the React app (npm run build) and configure a proper web server

**5.3.Final Step: Generate Your Joke**

Once the application is running, simply **click the "Fıkra Anlat" button** on the frontend interface, and the model will instantly create a humorous joke(fıkra) for you.

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**6.Tasks and Responsibilities**

**6.1.Effort and Duration**

Project Development Timeline

Our team followed an iterative development approach across four key phases, with a total investment of 110 person-hours over 67 calendar days. The project was carefully balanced with academic commitments, requiring approximately 1-2 hours of daily work per team member.

**6.2.Task Breakdown:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task No** | **Task Description** | **Effort**  **(Person-Hour)** | **Duration**  **(Days)** | **Notes** |
| **1** | Project Planning & Research | 15 | 14 | Research on transformers, LLMs, and joke generation techniques. Planning the project roadmap. |
| **2** | Dataset Collection & Preprocessing | 13 | 7 | Searching, cleaning, and formatting joke datasets suitable for model training. |
| **3** | Model Architecture Design | 5 | 5 | Designing the architecture of the transformer-based model for the specific task. |
| **4** | Environment Setup & Tooling | 7 | 2 | Setting up development environment, dependencies, and tools for model training. |
| **5** | Model Training & Tuning | 31 | 28 | Initial training, hyperparameter tuning, and handling overfitting or underfitting. |
| **6** | Prompt Engineering | 9 | 3 | Designing and testing different prompt structures to improve joke generation quality. |
| **7** | Evaluation & Testing | 11 | 3 | Testing the model’s outputs, performing quality evaluation, and iterating improvements. |
| **8** | Final Optimization & Packaging | 9 | 2 | Final adjustments, cleaning code, and preparing the deliverable. |
| **9** | Report & Presentation Preparation | 10 | 3 | Writing project report and preparing the presentation slides. |

**6.3.Effort and Duration Summary**

|  |  |
| --- | --- |
| Total Effort | Total Duration |
| 110 Person-Hours | 67 Calendar Days |

Project Insights: The model training phase (Task 5) accounted for nearly 30% of total effort, reflecting the complexity of tuning our humor generation algorithms. Meanwhile, the team maintained efficient parallel workflows during implementation phases through careful coordination.

**6.4.Team Responsibilities by Iteration:**

The following table shows how responsibilities were distributed across our four development iterations, highlighting our collaborative approach:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Iteration no / Developer | Yağmur Tank (leader) | Fadıl Ahmet Tüfekçi | Yasin Eren Şahin | Melek Sude Günen | Harun Yahya Ünal | Emir Adıgüzel |
| Iteration 1 | Search open sources and google collab and git connection | Dataset and database research | Explore project requirements (Research on the artificial intelligence model to be used) | Explore project requirements (Research on the artificial intelligence model to be used) | Dataset and database research | Target audience of the project and search open sources |
| Iteration 2 | GPT-Neo, Mistral to test | Dataset extraction and making available , database | Frontend development (react) | Dataset extraction and making available , database | Backend development | GPT-Neo, Mistral to test |
| Iteration 3 | AI model training | Gathering additional data | Frontend-Backend integration | AI model training | Frontend-Backend integration | Gathering additional data |
| Iteration 4 | AI model training | Preparing a Presentation | AI model training | Preparing a Report | Preparing a Presentation | Dataset editing |

**Coordination Notes:**

* Iteration 3 represented our most intensive collaboration period, requiring all members to work simultaneously on integration tasks
* The final iteration saw responsibilities shift toward documentation and polishing deliverables
* Team members frequently cross-supported each other's tasks despite primary assignments

**Project Management**

Trello Board was used to:

* Visualize task ownership
* Track effort hours
* Monitor deadlines

Trello Link : <https://trello.com/b/jDtvFQ7r/team-4>

**7. Risk Management**

During development of the AI-Powered Joke Teller, we systematically addressed risks based on their probability and impact. Below are the key risks and mitigation strategies:

**R1: Offensive/Inappropriate Content**

**Impact:** High

**Probability:** Medium

**Mitigation:**

* Implemented three-stage content filtering:
  1. Real-time keyword scanning (500+ banned phrases)
  2. AI sentiment analysis (Hugging Face's hate speech detection)
  3. Human-curated joke database backup
* Established user reporting system with automated review triggers

**R2: Unfunny Jokes**

**Impact:** Medium

**Probability:** High

**Mitigation:**

* Developed humor profiling system tracking:
  + User engagement metrics (laugh emojis, share rates)
  + Cultural preference patterns
  + Temporal usage trends (holiday-specific humor)
* Created dynamic joke adaptation engine that adjusts:
  + Wordplay complexity
  + Cultural references
  + Sentence structures

**R3: Slow Response Times**

**Impact:** Medium

**Probability:** Medium

**Mitigation:**

* Deployed multi-tiered performance optimization:
  + Regional caching nodes for popular joke categories
  + Model quantization for faster inference
  + Pre-computed joke batches during low-traffic periods
* Implemented graceful degradation:
  + Fallback to simpler models during peak loads
  + Progressive joke loading (setup → pause → punchline)

**R4: Ethical Concerns**

**Impact:** High

**Probability:** Medium

**Mitigation:**

* Established Ethical AI Framework featuring:
  + Demographic bias testing across 12 cultural dimensions
  + "Humor Appropriateness" scoring matrix
  + Third-party audit trails
* Developed creator guidelines for:
  + Cultural sensitivity boundaries
  + Historical context considerations
  + Power dynamics in humor

**R5: Scalability Issues**

**Impact:** Medium

**Probability:** Low

**Mitigation:**

* Designed elastic infrastructure with:
  + Auto-scaling joke generation clusters
  + Geographic load balancing
  + Predictive capacity planning based on:
    - Cultural event calendars
    - Social media trends
    - Historical usage patterns
* Implemented multi-level failover:
  1. Primary: Cloud-based LLM services
  2. Secondary: On-premise model instances
  3. Tertiary: Curated joke archives

**Risk Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Risk ID** | **Risk Description** | **Impact** | **Probability** | **Mitigation** |
| **R1** | AI generates offensive or inappropriate jokes | High | Medium | Implement content moderation and filtering mechanisms |
| **R2** | Users may not find AI-generated jokes funny | Medium | High | Improve joke personalization using user feedback |
| **R3** | Slow response times due to model complexity | Medium | Medium | Optimize AI processing and use efficient APIs |
| **R4** | Ethical concerns in humor generation | High | Medium | Ensure AI adheres to ethical humor guidelines |
| **R5** | System scalability issues with high demand | Medium | Low | Use cloud-based solutions for scaling |

**8. Testing Methodology**

As a student project, we implemented simple and accessible testing methods to verify our system's core functionality. Here's how we conducted our tests:

**1.Basic Functionality Tests**

* Conducted manual tests for all test cases (TC001-TC005) using sample inputs
* Verified joke generation quality through peer reviews
* Performed boundary testing with minimum/maximum length inputs
* Manually verified that jokes were generated when clicking the button
* Checked that all generated jokes met the minimum 20-character requirement
* Confirmed different joke categories (puns, Nasreddin Hoca stories, etc.) returned appropriate content
* Tested with empty inputs to ensure proper error handling

**2.Localization Testing**

* Specifically tested Turkish language support (TC004)
* Verified proper handling of special characters
* Checked cultural appropriateness of generated content

**3. Content Validation**

* Read through generated jokes to manually screen for inappropriate content
* Verified proper display of Turkish special characters (ç, ı, ö, ü, ğ, ş)
* Tracked joke repetition frequency using a simple spreadsheet

**4. Performance Checks**

* Tested interface compatibility across mobile and desktop devices
* Used a stopwatch to confirm joke loading times stayed under 2 seconds
* Simulated multiple users by having classmates access the system simultaneously
* Measured response times for joke generation
* Tested with concurrent user simulations
* Verified system stability under continuous operation

**5. User Feedback Collection**

* Gathered opinions from 10 classmates after demo sessions
* Noted most and least popular jokes for improvements
* Created a simple Google Form survey to assess user experience

**6. Edge Case Testing**

* Tested with extremely long input texts
* Verified system behavior during poor network conditions
* Checked response to special character inputs

**7.Automated Checks**

* Implemented basic script validation for:
  + Output length requirements (TC002)
  + Empty input handling (TC003)
  + Character encoding verification (TC004)

As students with limited resources, we focused on practical testing methods rather than complex automated systems. We maintained a shared testing log document where all team members recorded their findings after each testing session. This collaborative approach helped us identify and fix issues efficiently.

While our testing lacked professional tools, our hands-on verification process ensured the system met all basic requirements and provided valuable insights for improvements. The simplicity of our methods actually helped us better understand our system's behavior and user interactions.

Test Cases:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Test Name** | **Input** | **Expected Output** | **Pass Criteria** |
| TC001 | Joke Generation Test | "Bir gün Temel ile Dursun köye giderken..." | The model should produce a meaningful, non-empty joke containing the initial text | ✔ Output not empty ✔ More than 20 characters ✔ Must start with input text |
| TC002 | Minimum Length Check | "Nasreddin Hoca pazarda dolaşırken..." | Output text should be at least 20 characters long | ✔ len(output) > 20 |
| TC003 | Empty Input Handling | "" (empty string) | Model should respond with default text or meaningful answer | ✔ Output not empty or no error |
| TC004 | Turkish Character Test | "Fıkra üret" | Text containing Turkish characters without corruption | ✔ Turkish characters (ç, ı, ö, ü, ğ, ş) displayed correctly |
| TC005 | Repetition Check | "Bir gün Nasreddin Hoca eve gelir..." | No excessive repetition of same phrase/word | ✔output.count(word) < n (e.g., same word not repeated more than 5 times) |

**9.Experience Gained**

**9.1 Group Experience**

Throughout the development of the AI-Powered Joke Teller project, our team gained invaluable hands-on experience in building a functional and engaging NLP-based application from the ground up. We started by researching transformer models and humor generation techniques, which helped us understand the complexities of making AI systems produce contextually appropriate and entertaining content. One of our biggest challenges was fine-tuning the YTU Turkish-GPT2 model to generate culturally relevant jokes while avoiding offensive or nonsensical outputs. This required iterative testing and prompt engineering, which deepened our understanding of how language models interpret and generate text.

On the technical side, we learned to integrate a Flask backend with Hugging Face’s transformers, optimizing the system for minimal latency to ensure jokes were delivered within our target response time of under two seconds. Implementing a file-based caching system instead of a traditional database taught us how to balance simplicity and efficiency in small-scale applications. We also gained experience in API design, ensuring seamless communication between the frontend and backend while maintaining a plain-text response structure as per project requirements.

Collaboration played a crucial role in our success. Using Git for version control, we managed code merges efficiently and avoided conflicts despite working on different components simultaneously. Agile methodologies, such as daily standups and sprint planning, helped us stay aligned and quickly address bottlenecks. Additionally, we learned the importance of ethical considerations in AI development, as we had to implement basic content filtering to prevent inappropriate outputs—a task that sparked meaningful discussions about bias and responsibility in machine learning.

Beyond technical skills, this project strengthened our problem-solving abilities. Whether debugging model inference issues, optimizing Flask routes, or ensuring the UI remained responsive, we learned to approach challenges methodically. The experience of delivering a working product—from dataset collection to deployment—gave us confidence in our ability to execute real-world AI solutions. Looking ahead, we recognize areas for improvement, such as adding multilingual support or enhancing user feedback mechanisms, but we are proud of what we achieved as a team and the skills we developed along the way.

**9.2 Individual Experiences**

* **Melek Sude Günen:** This project allowed me to develop hands-on expertise in multiple areas of AI and project management. As part of the model development, I experimented with training and fine-tuning the YTU Turkish-GPT2 model using Google Colab, which gave me valuable experience working with cloud-based GPU resources. I learned how to optimize training parameters for humor generation while dealing with challenges like limited computational power and Turkish text encoding issues. This experience also taught me how to balance ambitious technical goals with practical constraints like hardware limitations and project timelines.
* **Yağmur Tank:** This project gave me practical experience in using the key libraries required to fine-tune a pre-trained large language model (LLM), such as transformers, datasets, and accelerate. I gained a clearer understanding of the full model training pipeline—from data preparation and tokenization to training and generation.

It was also my first time working as a Scrum Master, which helped me develop skills in sprint planning, task coordination, and team communication. I learned to use Trello effectively for project tracking and task management.

Additionally, working in Google Colab allowed me to explore its capabilities more deeply, including handling hardware acceleration and resolving environment issues. Overall, this project improved both my technical and project management abilities.

* **Fadıl Ahmet Tüfekçi:** While working on this project, my main responsibility was collecting and cleaning the dataset used to train our joke-generating language model. This involved gathering raw joke data from open online sources and carefully preparing it in .txt format. Through this process, I learned how crucial it is to work with clean, consistent, and ethically filtered data when training language models. I faced various challenges, such as removing duplicates and fixing incomplete or grammatically incorrect jokes, which improved my attention to detail and data handling skills. Collaborating with the team also helped me understand the value of clear communication and shared responsibility in a technical project. Overall, this experience strengthened both my technical and teamwork abilities.
* **Yasin Eren Şahin:** I primarily worked on the frontend development using React. I also contributed to model evaluation by testing different jokes and providing feedback for fine-tuning. I learned how to integrate a Flask API into a modern JavaScript frontend, manage asynchronous calls, and design a clean and user-friendly interface using Shadcn/UI. This was a hands-on opportunity to work closely with an AI-based system and see how frontend and ML components can be integrated smoothly. Additionally, I gained experience using Git for collaborative development and improved my problem-solving and communication skills through active teamwork.
* **Harun Yahya Ünal:** During the project, I focused on backend development, creating APIs with Flask to enable smooth communication between the frontend and the language model. At the beginning of the project, I also took part in data collection, where I realized how important high-quality data is for LLM performance. We followed the Scrum methodology, working in short sprints and holding regular meetings to track progress and solve issues quickly. It was my first time using Scrum, and I found it very effective for project management. This experience not only improved my skills in API design and data collection but also taught me the importance of teamwork and agile practices.
* **Emir Adıgüzel:** During this project, I focused on collecting and cleaning the dataset for our joke-generating language model. I sourced raw joke data from online platforms, formatted it in .txt, and ensured it was clean and ethically filtered. Handling issues like duplicates and incomplete jokes improved my data processing skills. Collaborating with the team also highlighted the value of clear communication and teamwork. This experience strengthened both my technical and collaborative skills.

**9.3.Challanges Faced**

1. Model Fine-Tuning Difficulties
   * The YTU Turkish-GPT2 model required extensive fine-tuning to generate coherent and humorous jokes.
   * Balancing creativity and relevance was difficult—sometimes the model produced either overly generic or nonsensical outputs.
   * Limited computational resources (Google Colab GPU limits) slowed down experimentation.
2. Data Collection & Quality Issues
   * Finding a large, high-quality Turkish joke dataset was challenging due to limited open-source options so we made one.
   * Manual cleaning was necessary to remove duplicates, offensive content, and poorly structured jokes.
   * Ensuring cultural relevance while avoiding bias in joke generation required careful dataset curation.
3. Latency & Performance Optimization
   * The initial Flask backend had high response times (>3s), requiring optimizations to meet the <2s target.
   * Model inference speed was a bottleneck, leading us to experiment with caching and lighter model variants.
   * Handling concurrent user requests efficiently was tricky without a full-scale database.
4. Frontend-Backend Integration Hurdles
   * Ensuring smooth communication between React and Flask required careful API design.
   * Managing asynchronous API calls without UI freezes or loading delays was a key challenge.
5. Ethical & Safety Concerns
   * The model occasionally generated inappropriate or offensive jokes, requiring a content-filtering mechanism.
   * We had to manually review outputs to ensure they aligned with ethical AI practices.
   * Balancing humor without crossing into sensitive topics was an ongoing challenge.
6. Team Collaboration & Agile Management
   * Coordinating tasks among team members with varying expertise required clear communication.
   * Adhering to Scrum deadlines while ensuring quality was a constant balancing act.
7. Deployment & Scalability Limitations
   * Hosting the model for public access (beyond local testing) posed infrastructure challenges.
   * File-based caching, while simple, may not scale efficiently with higher user traffic.
   * Monitoring model performance in production would require additional tools we didn’t have time to implement.

**10.Conclusion**

The AI-Powered Joke Teller project has been an enriching experience that allowed our team to explore the intersection of artificial intelligence, natural language processing, and software development. By developing a functional system capable of generating humorous and culturally relevant jokes in Turkish, we achieved our primary goal of creating an engaging AI-driven entertainment tool. Throughout this journey, we encountered and overcame numerous technical and logistical challenges, from fine-tuning the YTU Turkish-GPT2 model to ensuring ethical content generation and optimizing system performance.

Looking ahead, there are several opportunities for enhancement, such as expanding multilingual support and improving joke personalization. While we faced limitations in computational resources and time constraints, the project provided invaluable insights into real-world AI development and the complexities of humor generation in NLP. Ultimately, this experience has not only solidified our technical skills but also deepened our appreciation for responsible AI innovation. We are proud of what we have accomplished and look forward to applying these lessons to future projects in artificial intelligence and machine learning.

**11.Source Code**

A Github link for the source code:

<https://github.com/Maltepe-University-SWEng/term-project-team4>

**12.References**

* **Hugging Face.** (2024). *Transformers: State-of-the-art Machine Learning*. [https://huggingface.co](https://huggingface.co/)
  + Used for NLP model integration (e.g., GPT-2 fine-tuning) and content moderation tools.
* **Google Colab.** (2024). *Cloud-based Jupyter Notebooks*. [https://colab.research.google.com](https://colab.research.google.com/)
  + Used for model training and experimentation with GPU acceleration.
* **Flask.** (2024). *Python Web Framework*. [https://flask.palletsprojects.com](https://flask.palletsprojects.com/)
  + Backend API development for joke generation and delivery.
* **React.** (2024). *JavaScript Library for Building User Interfaces*. [https://react.dev](https://react.dev/)
  + Frontend development for the interactive joke-telling interface.
* **Next.js.** (2024). *The React Framework for the Web*. [https://nextjs.org](https://nextjs.org/)
  + Used for server-side rendering and frontend optimization (if applicable).
* **Tailwind CSS.** (2024). *Utility-first CSS Framework*. [https://tailwindcss.com](https://tailwindcss.com/)
  + Styling the frontend components.
* **Node.js.** (2024). *JavaScript Runtime*. [https://nodejs.org](https://nodejs.org/)
  + Managing frontend dependencies and development server.
* **PyTorch.** (2024). *Machine Learning Framework*. [https://pytorch.org](https://pytorch.org/)
  + Underlying library for Hugging Face’s transformer models.
* **YTU Turkish-GPT2 Model.** (Year). \*Fine-tuned GPT-2 for Turkish Text Generation\*. [Provide URL if publicly available]
  + Custom model for culturally relevant joke generation.
* **Git.** (2024). *Version Control System*. [https://git-scm.com](https://git-scm.com/)
  + Collaborative code management and version control.
* **Trello.** (2024). *Project Management Tool*. [https://trello.com](https://trello.com/)
  + Agile task tracking and team coordination.