Report for LLM Project

Prepared for
Ensar GÜL
Software Project Management
Maltepe University

Prepared by

Muharrem Şimşek – 210706001
Taha Demirhan – 210706002
Simay Aydın – 210706040
Aslımay Mısra Kandar – 210706043
Gökhan Mert Demirok – 220706311

Date
09.05.25

Contents

1.Scope of the Project

The **Joke-Telling AI** project aims to develop an artificial intelligence system capable of generating culturally appropriate, context-aware jokes interactively through a user-facing interface. Humor generation is a non-trivial problem in natural language processing (NLP) because it involves not just syntactic correctness but also semantic relevance, cultural nuance, timing, and creativity.

This project's primary objective is to design and implement a lightweight transformer-based language model, trained on a curated joke dataset, which can accept user input (typically a keyword or partial prompt) and complete it with a relevant, humorous joke. The system focuses on Turkish-language joke generation in this version, with future plans for multi-language support.

The project's **technical scope** includes:

- Fine-tuning a transformer decoder architecture for short-form joke generation.
- Training and integrating a custom SentencePiece tokenizer aligned with the model vocabulary.
- Developing a Streamlit-based web interface that allows seamless user interaction with the AI model.
- Managing data preprocessing, including cleaning, normalizing, and formatting the joke dataset for training efficiency.
- Implementing postprocessing techniques to ensure the generated output is coherent, culturally safe, and avoids nonsensical artifacts.

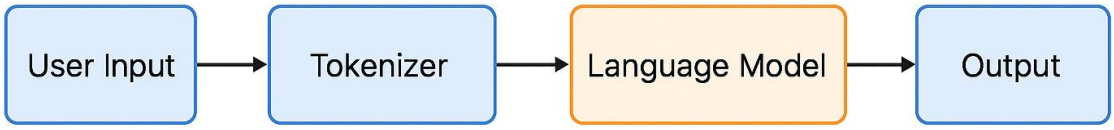
The project's **functional scope** is intentionally kept focused on core deliverables to ensure reliable, high-quality output within the project timeline. Specifically, this includes generating one-line or short jokes on-demand, interacting through a web interface, and providing interactive, turn-based chat history that preserves the conversational context.

Out-of-scope items (future work) include adding advanced personalization, implementing user rating feedback loops, and introducing speech or multimodal (audio/visual) joke delivery. These were identified as valuable but intentionally deferred to ensure a solid foundational implementation first.

Overall, the Joke-Telling AI project showcases not only technical implementation skills in NLP and AI but also addresses the unique challenges of embedding humor into artificial systems, balancing creativity with control, and ensuring ethical and culturally sensitive deployment.

2.Functionality of the Project

The **Joke-Telling AI** project offers a focused yet sophisticated set of functionalities designed to deliver an engaging user experience while demonstrating the power of transformer-based natural language models.



At its core, the project combines advanced machine learning components with an intuitive web interface to provide users with an interactive joke generation system. Below is a detailed overview of its key functional elements:

Joke Generation Engine

The system's primary function is to generate jokes based on user-provided prompts. The transformer decoder architecture—custom-built in PyTorch—accepts tokenized input and sequentially predicts the next tokens to form a coherent, humorous output. This autoregressive decoding process is powered by:

- Learned positional encodings, allowing the model to capture word order and context.
- A custom-trained SentencePiece tokenizer that aligns with the model's 10,000-word vocabulary, ensuring consistent tokenization and detokenization.
- Temperature-controlled sampling to balance between creativity and coherence during generation.

Interactive User Interface (Streamlit App)

The frontend, developed in Streamlit, offers an accessible and visually clean interface where users can:

- Input joke prompts or keywords directly into a text field.

- Receive real-time AI-generated joke responses displayed in a conversational format (chat history).
- Interact repeatedly, with session memory preserving prior exchanges for context continuity.
- The interface is lightweight, requiring no local installation beyond the necessary Python environment, and runs directly in a web browser.

Text Postprocessing

To improve user experience and output quality, the system includes a postprocessing pipeline that:

- Removes unwanted or meaningless character artifacts (e.g., repeated “??” or “??” symbols sometimes generated during decoding).
- Trims incomplete or incoherent endings by enforcing <EOS> token boundaries or punctuation-based stopping criteria.
- Ensures that jokes terminate cleanly with natural sentence boundaries.

Backend Model Management

The app is connected to a pre-trained model checkpoint (model_v11.pth), which loads with custom key remapping to align saved weights with the current model structure. This ensures that:

- The deployed system can be run smoothly on both CPU and GPU environments.
- Memory management is optimized, with efficient device loading depending on hardware availability.
- Future retraining or fine-tuning can be incorporated without major architectural rewrites.

Scalability for Future Features

Although the current version is focused on Turkish-language joke generation, the system is designed with modularity in mind, making it extendable to:

- Multi-language support (e.g., English, German) by training on additional datasets.
- Integration with larger conversational platforms like Telegram bots, mobile apps, or web plugins.
- Enabling rating mechanisms, allowing users to provide feedback on joke quality, which can be used for iterative model improvement.

a. Functions Available at the Beginning of the Project

At the project's start, the focus was solely on building a Turkish-language LLM (large language model) for joke generation. The team worked on:
• Developing the core transformer decoder architecture in PyTorch.
• Preparing the custom tokenizer aligned with the joke dataset.
• Training the initial model and testing it via scripts without any interface.

b. Functions Added During the Development

As development progressed, the team expanded the system's functionality significantly:
• Fine-tuning the model to improve humor quality and reduce errors.

Comparison: Pretrained vs. Fine-Tuned Model

	Pretrained Model	Fine-Tuned Model
Training Data	Broad and loosely cleaned dataset of jokes	Cleaned, normalized dataset with <EOS> tokens
Punchline Relevance	Often vague, off-topic, or abruptly ended	On-topic, culturally relevant, clear punchline endings
Output Length Control	Sometimes too short or too long	Better control via <EOS> and decoding strategy
Cultural Coherence (Turkish)	Mixed-quality jokes: sometimes English-inspired	Fluent and idiomatic Turkish humor structure
User Perception	Lower subjective ratings (less funny or confusing)	Cleaner raw output, fewer artifact characters (e.g., "?")

• Building an interactive Streamlit interface to let users input prompts easily and see responses in real time.
• Adding a postprocessing pipeline to clean up raw outputs from the model.
• Comparing model performance across versions (pre-finetuning vs. post-finetuning) to validate improvements.
• Preparing for future scalability, including modular architecture and hardware adaptability.

3.Missing Parts

Although the **Joke-Telling AI** project achieved its core functional goals, several planned or desirable features were ultimately left unimplemented, primarily due to time constraints, technical limitations, or project scope boundaries. Below is a detailed list of these missing parts and the reasons they were excluded:

Multi-language Support

The initial project vision included scaling the joke generator to handle multiple languages (e.g., English, German, French). However, due to the additional dataset requirements, separate tokenizers, and the need for multi-lingual model fine-tuning, this feature was postponed for a future version.

Joke Rating System

An interactive rating mechanism where users could score jokes (e.g., funny, neutral, unfunny) was part of the early design ideas. Such feedback could have been used for iterative model improvement, either by retraining or by fine-tuning joke selection strategies. This feature was not implemented due to the need for additional backend logic, user data handling, and longer development time.

Advanced Content Filtering

While the current system uses a cleaned and curated dataset, it does not include a dynamic, real-time content filter for sensitive, offensive, or culturally inappropriate jokes. Implementing such a filter would require integrating sentiment analysis, topic detection, or rule-based moderation layers—an advanced scope that was marked as future work.

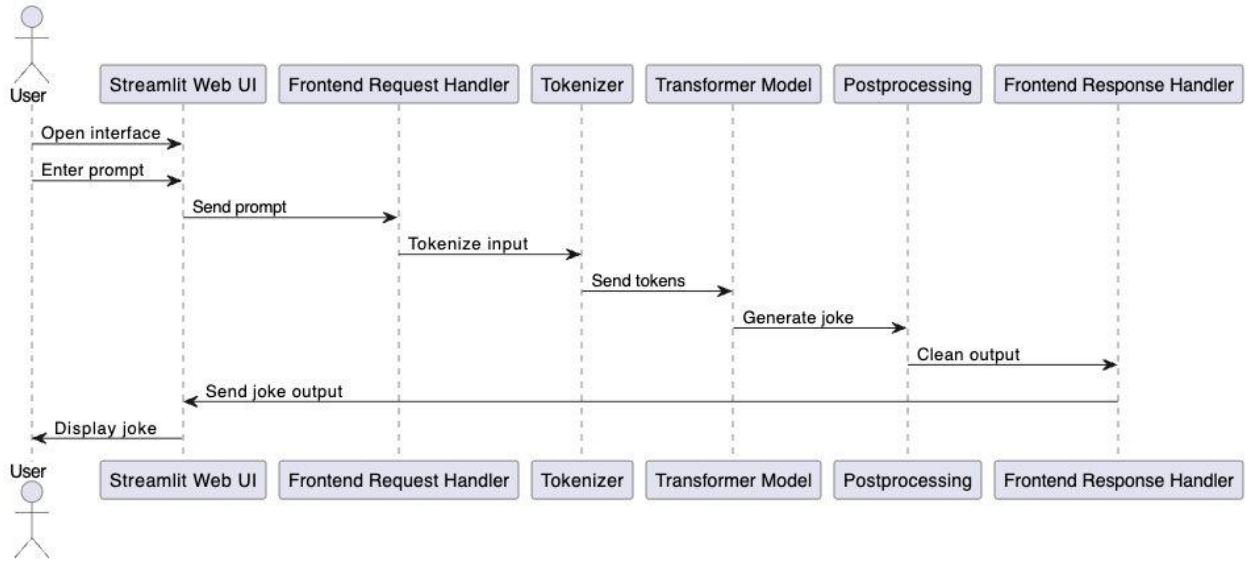
Personalization and User Profiling

The idea of tailoring jokes to user profiles, including humor preferences or interaction history, was discussed but deliberately left out to maintain focus on delivering a stable, generalized first version. Adding personalization would require user account management, preference tracking, and adaptive model behavior, which exceeded the current project scope.

Voice or Multimodal Interaction

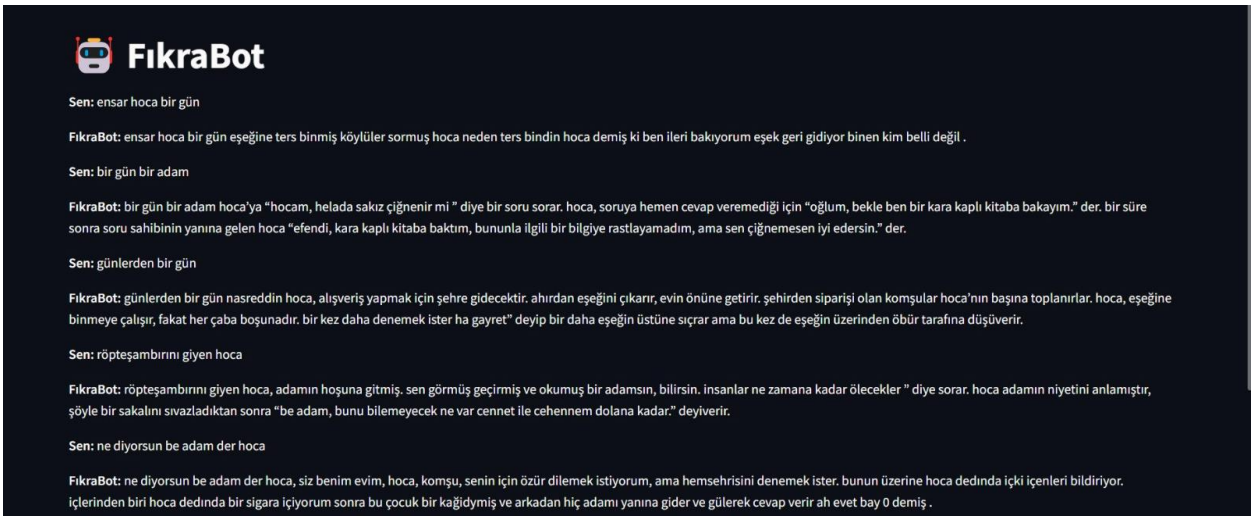
Although adding speech-based or multimodal (e.g., image-based humor) interfaces was seen as a potential enhancement, the team decided to concentrate on a text-only web interface to ensure focus and technical feasibility within the limited timeline.

4.Design documents

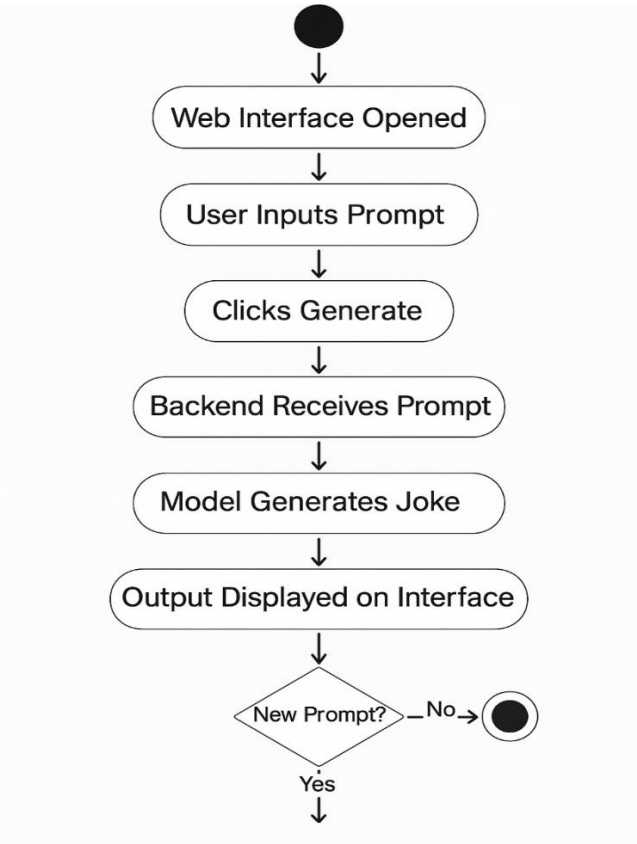


System Sequence Diagram: Shows the step-by-step message flow between the user, frontend, and backend components during a joke generation request.

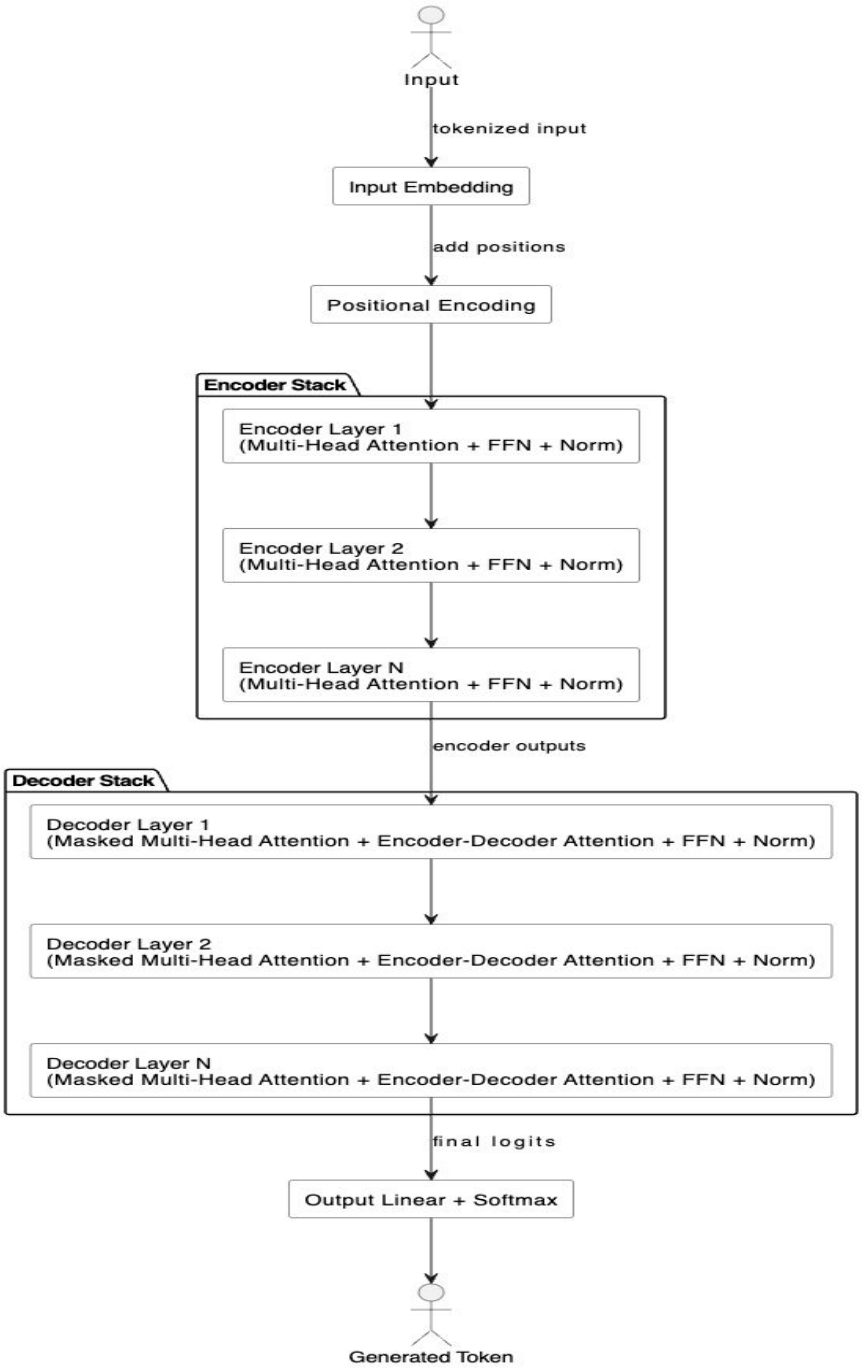
Screenshot of the Streamlit web application interface where users input prompts.



Screenshot of the Streamlit web application interface where users input prompts and receive AI-generated jokes in a conversational display.



System Flowchart: Visual representation of the interaction flow from the user interface to the backend model and back to the displayed output.



System Architecture Diagram: This diagram illustrates the internal architecture of a transformer model, showing the flow from input tokens through the encoder-decoder layers to the generated output token.

5. Deployment

The deployment process for the **Joke-Telling AI** system is designed to be lightweight, efficient, and accessible, leveraging Python's ecosystem and Streamlit's simplicity. Below is a step-by-step explanation of how to set up, run, and deploy the software:

System Requirements

- Python 3.8 or later
- PyTorch (compatible with local hardware: CPU or GPU)
- SentencePiece library
- Streamlit
- Required model files:
    - model_v11.pth (pre-trained language model weights)
    - fikra_tokenizer.model (custom tokenizer)

Local Deployment Steps

1.Clone or download the project repository to your local machine.

2.Ensure all dependencies are installed:

bash
pip install torch sentencepiece streamlit

3.Place the app.py, model checkpoint, and tokenizer file in the same working directory.

4.Run the Streamlit app with the following command:

bash
streamlit run app.py

5.Open the local browser at the URL provided by Streamlit to interact with the app.

Cloud Deployment (Optional)
Although the project is designed for local execution, it can be adapted for cloud deployment using platforms such as:

- **Streamlit Cloud (formerly Streamlit Sharing):** Deploy directly by pushing the code to a connected GitHub repository.
- **Heroku or Render:** Set up a Python web app, configure necessary buildpacks, and ensure persistent storage of model files.
-
- **Dockerized Deployment (Future Work):** Package the app and its dependencies into a Docker container for reproducible, scalable deployment.

Deployment Considerations

- **Hardware Utilization:** If GPU resources are available (local or cloud), PyTorch will automatically detect and use them, improving inference speed.
- **Scalability:** The current system is intended as a single-user or small-group demo. For production use, load balancing, persistent backend services, and a database layer would be required.
- **Security:** As the app does not currently handle user authentication or sensitive data, it is suitable for open demo environments. Future expansions may require secure endpoints and data protection measures.

6. Tasks and Responsibilities

Iter No / Developer	Muharrem	Mert	Taha	Mısra	Simay
Week 1	Determining and analyzing project requirements	Collecting necessary data and conducting research for the project	Determining and analyzing project requirements	Conducting necessary research and collecting data	Researching data collection techniques and gathering data
Week 2	Researching to build the LLM model, reviewing example models	Completing data collection for the model, planning and distributing tasks for the interface	Preparing APIs for model services and providing endpoints to the interface	Checking and fixing errors in the data after model creation and testing	Fixing the errors that appeared in the collected data
Week 3	Testing the model and fine-tuning	Starting to build the interface	Integrating model APIs with the interface and testing interactions	Providing support on the interface and additional data collection	Starting the interface design
Final Week	Final optimization and checking of the model	Preparing the project report	Testing the model, analyzing gaps, and preparing the Test Case report	Preparing the project presentation slides	Preparing the project presentation

Trello link: https://trello.com/b/9CScpG1H/team-1

7. Risk Management

During the development of the **Joke-Telling AI** project, the team encountered several technical, operational, and ethical risks. Below is an overview of the identified risks, their potential impact, and the strategies used to mitigate them:

---

1. Subjectivity of Humor

- **Risk:** Humor is inherently subjective and culturally dependent. Jokes that are funny to one group may be perceived as flat, offensive, or nonsensical to another.
- **Mitigation:**
    - Use of a curated, pre-filtered Turkish joke dataset.
    - Focus on simple, universal humor types (such as puns and one-liners) rather than complex cultural references.
    - Future plan: introduce user feedback loops to improve joke relevance over time.

2. Data Quality and Consistency

- **Risk:** Raw joke data contained inconsistencies, noise, and formatting issues that could negatively impact model training and output quality.
- **Mitigation:**
    - Manual data cleaning and normalization (removing punctuation, lowercasing, ensuring proper <EOS> tagging).
    - Validation steps to check dataset integrity before training.
    - Postprocessing of generated outputs to remove or correct garbled text.

---

3. Technical Limitations and Model Instability

- **Risk:** The transformer-based language model could produce incoherent or incomplete outputs, especially when prompted with vague or out-of-domain inputs.
- **Mitigation:**
    - Controlled prompt design and prompt length restrictions.
    - Implementation of temperature-controlled sampling to balance creativity and stability.
    - Testing with diverse prompt sets to evaluate model robustness.

---

4. Ethical and Cultural Risks

- **Risk:** Generated jokes might unintentionally reflect biases, stereotypes, or culturally inappropriate themes inherited from the training data.
- **Mitigation:**
  - Careful dataset curation to remove sensitive or potentially offensive material.
  - Ethical review of model outputs during testing.
  - Clear documentation of system limitations and ethical boundaries in the project report.

---

5. Project Timeline and Resource Constraints

- **Risk:** Limited development time and human resources posed challenges for completing all planned features and conducting extensive testing.
- **Mitigation:**
  - Prioritization of core features (minimum viable product).
  - Task delegation across team members according to skill sets.
  - Weekly progress check-ins to adjust plans and mitigate delays.

8. Tests

To ensure the **Joke-Telling AI** system operated as intended, the team implemented multiple testing strategies covering both technical performance and user experience. Below is a breakdown of the testing methods used:

---

1. Functional Testing

- **Goal:** Verify that each system component (tokenizer, model, postprocessing, frontend interface) worked correctly on its own and when integrated.
- **Approach:**
  - Manual testing of tokenization: Checking that input text was correctly split into tokens and reconstructed after detokenization.
  - Model output checks: Ensuring that, given a prompt, the model generated grammatically and semantically coherent joke completions.
  - Frontend testing: Testing input fields, chat display, and session history management in the Streamlit app.

2. Integration Testing

- **Goal:** Ensure smooth interaction between the frontend (Streamlit app) and backend (PyTorch model and tokenizer).
- **Approach:**
    - End-to-end tests where a prompt was entered in the interface, passed to the backend, and returned as a formatted joke to the user.
    - Checking for consistent response times and ensuring no crashes or unexpected behaviors under normal loads.

---

3. User Testing

- **Goal:** Evaluate how users perceived the quality and humor of the generated jokes.
- **Approach:**
    - Team members and external testers provided prompts and rated the resulting jokes as “funny,” “acceptable,” or “not funny.”
    - Collected informal feedback on joke relevance, coherence, and entertainment value.

---

4. Error Handling and Stability Testing

- **Goal:** Test how the system handled invalid inputs, extreme prompts, or unexpected behaviors.
- **Approach:**
    - Input edge cases: Empty prompts, excessively long strings, or special characters.
    - Observed how gracefully the system handled errors (e.g., returning a default message or skipping malformed requests).
    - Ensured that the app did not break or freeze under multiple, rapid requests.

---

5. Tools and Frameworks Used

- Manual script-based testing for backend components.
- Streamlit’s local testing environment for frontend validation.
- No formal automated testing frameworks were implemented due to project scope, but basic unit and integration tests were performed manually.

9. Experience Gained

This project provided valuable technical, organizational, and teamwork experiences for each group member. Below are individual summaries of the key learnings and skills gained:

Muharrem
I deepened my understanding of transformer-based language models and learned how to fine-tune them for domain-specific tasks like humor generation. I gained hands-on experience integrating backend AI models with frontend applications using Streamlit and learned how to perform final model optimization and testing to ensure stable system performance.

Mert
I strengthened my skills in data collection and preparation, learning how to curate a domain-specific dataset for natural language processing tasks. I also gained experience designing and implementing the user interface, understanding the practical considerations when connecting the frontend to backend AI services.

Taha
I learned how to design and implement API services that connect machine learning models to user interfaces, including writing API endpoints and testing model interactions. Additionally, I developed skills in writing structured Test Case reports and performing systematic error analysis to improve the model’s performance.

Mısra
I gained valuable experience in data cleaning and quality control, learning how even small inconsistencies can impact AI model behavior. I also contributed to supporting the frontend integration and preparing the project presentation materials, which helped strengthen my technical communication and collaboration skills.

Simay
I focused on researching data collection techniques and gained practical experience designing user interfaces that enhance interaction and usability. Additionally, I contributed to preparing the project’s final presentation, learning how to translate complex technical work into clear, accessible materials for diverse audiences.

10. Source Code Repository

Github Link: https://github.com/Maltepe-University-SWEng/term-project-team-1

11. References

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017).
Attention is All You Need.
In *Advances in Neural Information Processing Systems (NeurIPS)*.
https://arxiv.org/abs/1706.03762

PyTorch.
An open-source machine learning framework.
Available at: https://pytorch.org/

Streamlit.
The fastest way to build and share data apps.
Available at: https://streamlit.io/

Google SentencePiece.
Unsupervised text tokenizer and detokenizer for Neural Network-based text generation.
Available at: https://github.com/google/sentencepiece

Hugging Face Transformers.
State-of-the-art Natural Language Processing for Pytorch and TensorFlow.
Available at: https://huggingface.co/transformers/

Google Colab.
Collaborative Python notebooks in the cloud.
Available at: https://colab.research.google.com/