VISION AND SCOPE DOCUMENT:

Developing LLM, which generates Turkish Jokes

Project Details	
University	Maltepe University
Course	Software Project Management
Teacher	Ensar Gül

Project Participants	
Full Name	**Student Number**
Alp Salcıoğlu	220706016
Berke Ensel	210706039
İbrahim Kartal	210706004
Esma Şen	
Melahat Eda Acar	
Eren Yıldırım	

Table Of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to define the vision and scope of the **Turkish Joke Generating LLM** project. This document serves as a guide for all stakeholders, including the development team, project manager, and end users, to ensure a clear understanding of the project's objectives, functionalities, and constraints.

The project aims to develop a **small-scale Large Language Model (LLM) capable of generating Turkish jokes** based on user input. By leveraging **PyTorch**, the model will be trained from scratch using a dataset of **approximately 1,500 jokes**. The expected outcome is a model that produces contextually relevant and humorous responses, improving over multiple iterations.

This document will outline the high-level requirements, system architecture, constraints, and risks associated with the project. Additionally, it will provide an overview of the project's stakeholders and deliverables in alignment with the Scrum methodology.

1.2 Scope of the Project

The **Turkish Joke Generating LLM** project is designed to explore the capabilities of **natural language processing (NLP) and deep learning** in generating humorous text. The system will accept a **text-based user input (prompt) and generate a joke as output**.

The key aspects of the project scope include:

- **Model Development**: A PyTorch-based LLM trained on Turkish jokes.
- **Dataset**: A collection of **1,500 Turkish jokes** that will be cleaned and preprocessed.
- **Training & Evaluation**: The model will be trained for **approximately 100 epochs**, with performance evaluated based on joke coherence and humor quality.
- **User Interaction**: A simple **command-line or web-based interface** (optional) for testing joke generation.
- **Project Methodology**: The team follows the **Scrum framework**, with iterative improvements across sprints.

Out of Scope:
The following aspects are not within the project's scope:

- **Advanced LLM capabilities** beyond joke generation.
- **Integration with external APIs or chatbot frameworks** (e.g., OpenAI API).
- **Multilingual joke generation** (limited to Turkish only).

1.3 Objectives

The main objectives of this project are:

Objective	Description
Develop an LLM from scratch	Build a small-scale language model using PyTorch.
Train on Turkish joke dataset	Utilize a dataset of 1,500 jokes and optimize model performance.
Ensure joke relevance & coherence	Improve joke structure, fluency, and humor accuracy.
Implement iterative improvements	Apply Scrum methodology, refining the model in sprints.
Provide an interactive interface	Enable users to test joke generation (optional).

1.4 Definitions and Acronyms

Term	Definition
LLM (Large Language Model)	A deep learning-based model trained to understand and generate human-like text.
NLP (Natural Language Processing)	A field of AI focused on enabling machines to understand and generate human language.
PyTorch	An open-source deep learning framework used for training neural networks.
Epoch	A complete pass through the training dataset during model training.
Scrum	An Agile project management framework emphasizing iterative development.

2. System Overview

2.1 High-Level Description

The **Turkish Joke Generating LLM** project is designed to develop a small-scale **Large Language Model (LLM)** capable of generating Turkish jokes based on a given input. The system is built using **PyTorch**, and it is trained on a **dataset of 1,500 Turkish jokes**.

The model follows a **sequence-to-sequence (Seq2Seq) architecture** with a transformer-based encoder-decoder mechanism. The training process consists of multiple iterations (**approximately 100 epochs**) to refine joke coherence and humor.

The system consists of the following main components:

1. **Data Collection & Preprocessing** – Cleaning and structuring the joke dataset for training.

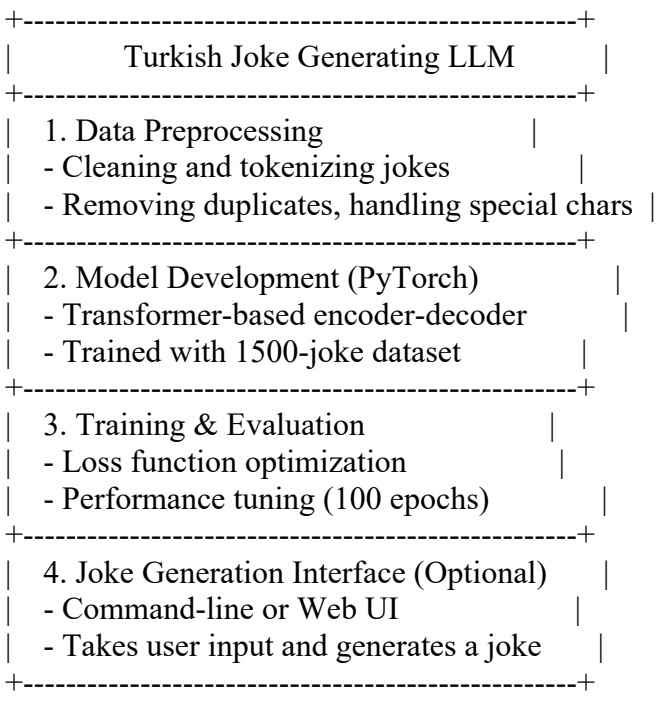
2. **Model Architecture** – Implementing and training a transformer-based model using PyTorch.
3. **Training & Fine-Tuning** – Optimizing the model's parameters to enhance joke generation.
4. **Evaluation & Testing** – Measuring performance using accuracy metrics and qualitative human evaluation.
5. **User Interaction (Optional)** – Providing a command-line or web-based interface for testing.

The final model will allow users to enter a text prompt, and the system will generate a joke that aligns with the given input.

2.2 System Architecture

The system follows a **modular architecture**, enabling flexibility in training, testing, and deployment. Below is a **high-level architectural diagram** of the system:

2.2.1 System Architecture Diagram



2.2.2 System Components

Component	Description
Data Preprocessing	Cleans and tokenizes joke dataset, removing irrelevant content.
Model Architecture	Uses a **Transformer-based LLM** trained on Turkish jokes.
Training Process	Optimizes model using **Adam optimizer** and **cross-entropy loss function** over 100 epochs.
Evaluation	Uses **BLEU score** and **human evaluation** for joke quality assessment.
User Interface **(Optional)**	Allows users to test joke generation via CLI or web UI.

2.3 System Workflow

The **workflow** of the system follows these steps:

1. **Data Collection & Cleaning**
    - Load the Turkish joke dataset.
    - Clean text, remove duplicates, and tokenize sentences.
2. **Model Training & Optimization**
    - Implement a Transformer-based model.
    - Train using PyTorch for 100 epochs.
    - Optimize using **Adam optimizer** and **cross-entropy loss function**.
3. **Joke Generation & Testing**
    - Input a text prompt.
    - Model generates a relevant joke.
    - Evaluate coherence using BLEU scores and human validation.
4. **Deployment & Interaction**
    - Provide a simple CLI/web UI for user testing.

3. Functional Requirements

3.1 Data Collection and Preprocessing

The system requires a **structured and cleaned dataset of Turkish jokes** to ensure high-quality training data. The preprocessing steps include:

- **Text Cleaning:** Removing special characters, redundant spaces, and irrelevant symbols.
- **Tokenization:** Splitting jokes into individual tokens for better model training.
- **Removing Duplicates:** Eliminating repeated jokes to maintain dataset diversity.
- **Handling Special Cases:** Addressing non-standard language elements (e.g., slang, abbreviations).

Step	Description
Dataset Collection	1,500 Turkish jokes gathered from various sources.
Data Cleaning	Removing punctuation, special characters, and HTML tags.
Tokenization	Splitting text into words or subwords.
Duplicate Removal	Filtering out repeated jokes.

3.2 Model Design

The model follows a **transformer-based sequence-to-sequence (Seq2Seq) architecture** designed using **PyTorch**. Key components include:

- **Input Layer:** Takes tokenized joke prompts as input.
- **Encoder-Decoder Architecture:** Transformer layers to generate meaningful jokes.
- **Training Pipeline:** Optimized with **Adam optimizer** and **cross-entropy loss function**.

Component	Description
Embedding Layer	Converts words into vector representations.
Encoder	Processes input and captures semantic meaning.
Decoder	Generates joke text based on encoded input.
Output Layer	Produces final generated joke.

3.3 Model Training and Evaluation

The model will be trained using **100 epochs** with periodic evaluation.

Training Process:

- **Loss Function:** Cross-entropy loss to measure prediction accuracy.
- **Optimizer:** Adam optimizer for better convergence.
- **Batch Size:** Optimal batch size selection for stable learning.
- **Epochs:** 100 iterations for improved joke coherence.

Evaluation Metrics:

- **BLEU Score:** Measures linguistic accuracy and fluency.
- **Perplexity:** Evaluates the model's ability to generate coherent text.
- **Human Validation:** Subjective assessment of joke quality.

Metric	Purpose
BLEU Score	Measures the similarity between generated and real jokes.
Perplexity	Evaluates how well the model predicts the next word.
Human Review	Assesses joke coherence and humor level.

3.4 User Interface (Optional)

A **simple CLI or web-based interface** may be implemented for user interaction.

Feature	Description
Text Input Field	Allows users to enter a joke prompt.
Generate Button	Triggers the model to produce a joke.
Joke Output Area	Displays the generated joke.

4. Functional Requirements

This section defines the functional requirements of the project, outlining the essential features and capabilities that the system must provide.

4.1 Input Processing

- The model should accept **text-based inputs** from users in natural language (Turkish).
- Input length should be dynamically adjustable but should not exceed **256 characters** to prevent unnecessary processing load.
- The system should be able to **preprocess inputs**, including:
    - Removing unnecessary spaces, special characters, and inappropriate words.
    - Converting text to lowercase if needed for uniformity.
    - Tokenizing input for proper model processing.

Vision Component	Description
Primary Goal	Develop a Turkish joke generation model using NLP techniques in PyTorch.
Target Audience	Turkish-speaking users looking for humorous content.
Core Functionality	Generate relevant and funny Turkish jokes based on user input.
Key Differentiator	A model specifically trained on Turkish jokes, focused on cultural relevance.
User Experience	Easy-to-use API for seamless integration into different platforms or applications.
Impact	Provide a lighthearted and culturally appropriate joke generation service.

4.2 Joke Generation Model

- The system should generate **coherent and contextually appropriate jokes** based on the given input.
- The model should produce **grammatically correct and semantically meaningful** responses.
- The generated jokes should be diverse and should avoid excessive repetition.
- The response should be generated within **2 seconds** for optimal user experience.
- The model should use **temperature sampling** or **top-k/top-p sampling** techniques to ensure variation in outputs.

4.3 Post-Processing

- The system should include a **content filtering mechanism** to detect and block inappropriate, offensive, or biased jokes.
- A rule-based approach and **machine learning-based toxicity detection** should be applied to eliminate undesirable outputs.
- The model should be able to re-generate responses if the output is flagged as inappropriate.

4.4 User Interaction and API Integration

- The model should be accessible via an **API endpoint**, allowing integration with external applications.
- The API should support:
    - POST requests with a JSON payload containing the input text.
    - GET requests for retrieving system status and logs.
- The API response should include:
    - The generated joke.
    - A confidence score for the output.
    - A response time metric.

4.5 Data Handling and Storage

- The system should log user queries and generated responses (while maintaining privacy and security).
- The logs should include:
    - Input text.
    - Output joke.
    - Processing time.
    - Possible flagged content (if any).
- The data should be stored securely, ensuring compliance with **data protection regulations**.

4.6 Performance and Scalability

- The system should support **at least 100 concurrent requests** without performance degradation.
- The model should be optimized to minimize **computational costs** while maintaining quality.
- The API should be designed to handle **increased traffic** efficiently with **scalable deployment strategies** (e.g., using containerized environments like Docker).

5. Non-Functional Requirements

5.1 Performance Requirements

- The model should generate a joke **within 2 seconds** after receiving input.
- The system should handle **at least 100 concurrent requests** without significant performance degradation.
- The response should maintain an accuracy rate of **at least 80%**, meaning that most generated jokes should be considered coherent and relevant.
- The model should have a **perplexity score lower than 50**, indicating good language understanding.

5.2 Scalability

- The system should be **scalable** to accommodate increasing requests without a significant drop in performance.
- A **load-balancing mechanism** should be implemented to distribute traffic efficiently.
- The model should support **deployment in a cloud-based environment** (e.g., AWS, Google Cloud, or Azure) for better scalability.

5.3 Security and Data Privacy

- The system must ensure **user data privacy** by not storing or logging personally identifiable information (PII).
- **Encrypted communication (HTTPS)** should be enforced for all API interactions.
- Access to the model and API should be protected by **authentication and authorization mechanisms**.
- The system should prevent **injection attacks** and ensure robustness against adversarial inputs.

5.4 Usability and User Experience

- The API should be **well-documented** to facilitate easy integration with external applications.
- The generated jokes should be **easy to read and understand**, ensuring a smooth user experience.
- The system should provide **meaningful error messages** in case of failures or invalid inputs.

5.5 Maintainability and Extensibility

- The codebase should follow **modular and clean coding practices** to allow future improvements.
- The system should support **future updates** with minimal disruption.
- A **logging and monitoring mechanism** should be implemented to track system performance and detect issues.

5.6 Compliance and Ethical Considerations

- The system must adhere to **ethical AI principles**, avoiding bias, discrimination, and offensive content.
- A **content moderation mechanism** should be in place to filter inappropriate jokes.
- The project should comply with **applicable AI and data regulations**, such as GDPR (if applicable).

6. Constraints and Assumptions

6.1 Constraints

The following constraints must be considered during the development of the system:

6.1.1 Data Constraints

- **Dataset Size**: The project is limited to a dataset of **1500 jokes**, which may restrict the diversity of the generated content. The model should work effectively within this constraint, and any extrapolation for generalization should be done carefully.
- **Data Quality**: The dataset contains **Turkish jokes**, and the model must be able to handle any language-specific peculiarities, including slang, cultural references, and humor nuances.
- **Data Preprocessing**: The data must undergo preprocessing to remove any potentially harmful or offensive content, which may reduce the overall size of the usable data.

• Constraint	Description
Training Dataset Size	Limited to 1500 Turkish jokes, which may impact model generalization.
Model Complexity	The model must be lightweight due to limited computational resources.
Hardware Availability	Local machine resources may limit training times and batch sizes.
API Performance	The API must respond within 2 seconds for a good user experience.

6.1.2 Computational Constraints

- **Hardware Limitations**: The development will be done on systems with limited GPU resources. Models must be optimized to work efficiently without excessive computational cost.
- **Training Time**: Due to hardware limitations, the model will be trained for **up to 100 epochs**, and model convergence should be achieved within this constraint.
- **Model Size**: The model should not exceed **5GB in size** to ensure it can be easily deployed and used in production environments.

6.1.3 Time Constraints

- **Project Timeline**: The project must be completed within the duration of the academic semester, which is **approximately 3 months**. This includes the training, testing, and evaluation phases.

- **Sprint Deadlines**: All features, testing, and documentation must be completed within the 2-week sprint cycles outlined in the project timeline.

6.1.4 Ethical Constraints

- **Content Moderation**: The system must filter out harmful, inappropriate, or offensive content. This includes the use of automated filtering tools to ensure that generated jokes align with ethical standards and do not perpetuate harmful stereotypes.
- **Bias and Fairness**: The model must be developed and tested to ensure it does not generate biased or unfair content based on gender, race, or other sensitive factors. A fairness evaluation should be included.

6.2 Assumptions

The following assumptions have been made for the successful development and implementation of the project:

6.2.1 Data Assumptions

- **Data Quality**: It is assumed that the jokes in the dataset are representative of the type of humor expected from the model, with no major bias or errors.
- **Sufficient Data Coverage**: It is assumed that the dataset is sufficiently diverse to generate a wide variety of jokes, and the jokes are balanced across different themes.

6.2.2 Technical Assumptions

- **Availability of Tools**: It is assumed that all necessary tools, including PyTorch, the required machine learning libraries, and computational resources, will be available and accessible throughout the project.
- **Model Convergence**: It is assumed that the model will converge within the provided number of epochs (100 epochs) and will reach an acceptable level of accuracy for joke generation.

6.2.3 User Assumptions

- **User Familiarity**: It is assumed that users interacting with the system will have basic knowledge of how to input text and interact with the API.
- **Feedback Mechanism**: It is assumed that users will provide feedback on the quality of generated jokes, which will help improve future versions of the model.

6.2.4 External Dependencies

- **Third-Party Services**: It is assumed that no critical external dependencies, such as third-party APIs or services, will be required for the core functionality of the model. The model should work independently of external services unless explicitly specified.

7. Risks and Mitigations

7.1 Risks

Risk	Impact	Likelihood	Mitigation Strategy
Data Quality Risk	Inaccurate or biased jokes that harm the system's reliability.	Medium	Regular data review and use of content moderation tools.
Model Generalization Risk	Model may not generate diverse humor.	High	Use data augmentation or additional datasets to diversify training.
Computational Resource Limitations	Slow training and resource bottlenecks.	High	Use cloud-based services, optimize model efficiency.
Ethical and Bias Risks	Unintended offensive or biased jokes.	Medium	Implement fairness tests and content moderation at every stage.
User Experience Risks	Poor joke quality leads to low user engagement.	Medium	Use user feedback to continuously improve the model's output.

7.1.1 Data Quality Risk

- **Description**: The dataset may contain biases, offensive content, or inaccuracies, which could affect the quality of the generated jokes.
- **Impact**: Poor data quality can result in inappropriate, irrelevant, or offensive joke generation.
- **Likelihood**: Medium.

7.1.2 Model Generalization Risk

- **Description**: With a small dataset of 1500 jokes, the model may not generalize well to all types of humor or contexts.
- **Impact**: The model may generate repetitive or overly specific jokes that fail to capture the full spectrum of humor.
- **Likelihood**: High.

7.1.3 Computational Resource Limitations

- **Description**: Limited computational resources could slow down model training or restrict the number of experiments that can be conducted.
- **Impact**: Extended training times or limited tuning and evaluation cycles could affect the model's performance.
- **Likelihood**: High.

7.1.4 Ethical and Bias Risks

- **Description**: The model may unintentionally generate biased or offensive jokes due to cultural biases present in the training data.
- **Impact**: The system may generate harmful or offensive content, damaging its credibility and violating ethical guidelines.
- **Likelihood**: Medium.

7.1.5 User Experience Risks

- **Description**: Users may find the jokes generated by the system inappropriate or unfunny, leading to poor user engagement.
- **Impact**: Lack of user engagement may lead to a failed product or poor adoption.
- **Likelihood**: Medium.

7.1.6 API Integration Issues

- **Description**: There may be challenges with the integration of the joke generation model via the API, such as connectivity or response time issues.
- **Impact**: Poor API performance can affect the reliability and user satisfaction with the product.
- **Likelihood**: Low.

7.2 Mitigation Strategies

7.2.1 Data Quality Mitigation

- **Strategy**: Regularly review and clean the dataset to remove any offensive or biased content. Use content moderation tools to filter inappropriate jokes.
- **Action**: Implement both rule-based and machine learning-based filters during the preprocessing stage.

7.2.2 Model Generalization Mitigation

- **Strategy**: Augment the dataset through synthetic data generation or by collecting additional jokes to improve model diversity.
- **Action**: Fine-tune the model regularly based on user feedback to capture a broader range of humor.

7.2.3 Computational Resource Mitigation

- **Strategy**: Optimize model architecture for efficient training and minimize resource usage. Consider using cloud-based services for training if local resources are insufficient.
- **Action**: Use techniques like model pruning, quantization, or mixed precision training to reduce computational load.

7.2.4 Ethical and Bias Mitigation

- **Strategy**: Implement a thorough evaluation of the model's output to detect and remove biased or offensive jokes.
- **Action**: Introduce fairness evaluation tests and apply additional layers of content filtering to improve the ethical standards of generated content.

7.2.5 User Experience Mitigation

- **Strategy**: Use feedback loops to gather user input on joke quality, and adapt the model based on the feedback received.
- **Action**: Conduct user testing sessions and iterate on joke quality improvement based on real-world feedback.

7.2.6 API Integration Mitigation

- **Strategy**: Perform thorough integration testing to ensure smooth API functionality. Monitor API performance regularly to address potential issues proactively.
- **Action**: Implement load testing and error-handling mechanisms to ensure high availability and reliability of the API.

8. Acceptance Criteria

Functional Criteria	Acceptance Criteria
Joke Generation	Must generate relevant Turkish jokes with coherence and humor.
API	The API must accept text input and return jokes in a JSON format.
Performance	The model should return jokes within 2 seconds and maintain 80% accuracy.
Content Moderation	Must filter out inappropriate or biased jokes using moderation tools.

8.1 Functional Criteria

8.1.1 Joke Generation

- The model must generate a **relevant Turkish joke** based on user input.
- The joke must be **coherent, humorous, and culturally appropriate**.
- The joke generation process should be **completed within 2 seconds** after the user submits input.

8.1.2 User Interface (API)

- The system must provide a **RESTful API** to allow users to interact with the joke generation model.
- The API should accept input in the form of **text** (e.g., user-provided prompts or a predefined context).
- The API should return a **JSON response** containing the generated joke.

8.1.3 Performance

- The system must maintain an **accuracy rate of at least 80%** for joke relevance and quality.
- The system should support **at least 100 concurrent requests** without significant performance degradation.

8.1.4 Content Moderation

- The system must **filter out harmful or offensive jokes** before presenting them to the user.
- The system should adhere to **ethical AI standards**, ensuring that the jokes do not perpetuate harmful stereotypes or biases.

8.2 Non-Functional Criteria

8.2.1 Scalability

- The system must be able to handle **increased load** by scaling horizontally (e.g., via cloud infrastructure) to accommodate growing numbers of requests.

8.2.2 Security

- The system must use **secure communication protocols** (HTTPS) for all interactions.
- The API should be protected by **authentication and authorization** mechanisms to ensure only authorized users can access the joke generation service.

8.2.3 Maintainability

- The system's codebase should be **modular, well-documented**, and easy to maintain.
- The project should follow **clean coding practices**, ensuring that future updates or changes can be implemented efficiently.

8.2.4 Availability

- The system must be available for use **24/7**, with minimal downtime for maintenance or updates.
- **Error handling** should be implemented to ensure that the system can recover gracefully from failures.

8.3 User Experience Criteria

8.3.1 Usability

- The system must be **easy to use**, with clear instructions for interacting with the API and submitting input.
- The jokes generated by the model should be **understandable and humorous** to the target audience.

8.3.2 Feedback Loop

- The system should provide users with the option to **rate the quality** of the jokes they receive, and this feedback should be used to improve the model.

Vision Component	Description
Primary Goal	Develop a Turkish joke generation model using NLP techniques in PyTorch.
Target Audience	Turkish-speaking users looking for humorous content.
Core Functionality	Generate relevant and funny Turkish jokes based on user input.
Key Differentiator	A model specifically trained on Turkish jokes, focused on cultural relevance.
User Experience	Easy-to-use API for seamless integration into different platforms or applications.
Impact	Provide a lighthearted and culturally appropriate joke generation service.

Constraint	Description
Training Dataset Size	Limited to 1500 Turkish jokes, which may impact model generalization.
Model Complexity	The model must be lightweight due to limited computational resources.
Hardware Availability	Local machine resources may limit training times and batch sizes.
API Performance	The API must respond within 2 seconds for a good user experience.

Risk	Impact	Likelihood	Mitigation Strategy
Data Quality Risk	Inaccurate or biased jokes that harm the system's reliability.	Medium	Regular data review and use of content moderation tools.
Model Generalization Risk	Model may not generate diverse humor.	High	Use data augmentation or additional datasets to diversify training.
Computational Resource Limitations	Slow training and resource bottlenecks.	High	Use cloud-based services, optimize model efficiency.
Ethical and Bias Risks	Unintended offensive or biased jokes.	Medium	Implement fairness tests and content moderation at every stage.
User Experience Risks	Poor joke quality leads to low user engagement.	Medium	Use user feedback to continuously improve the model's output.

Functional Criteria	Acceptance Criteria
Joke Generation	Must generate relevant Turkish jokes with coherence and humor.
API	The API must accept text input and return jokes in a JSON format.
Performance	The model should return jokes within 2 seconds and maintain 80% accuracy.
Content Moderation	Must filter out inappropriate or biased jokes using moderation tools.

Term	Definition
Joke Generation Model	A model trained to generate Turkish jokes using NLP techniques.
Dataset	A collection of 1500 Turkish jokes for training the joke generation model.
API	An interface for interacting with the joke generation model.
Scrum Methodology	An agile framework for managing the development process.
Epoch	One complete pass through the training dataset during model training.
Content Moderation	The process of filtering out harmful, offensive, or biased content.
Natural Language Processing (NLP)	Techniques for enabling machines to understand and generate human language.
Accuracy	The percentage of jokes that meet the relevance and quality criteria.
Model Pruning	Reducing the size of the model by removing weights that have little effect.
Fine-tuning	Adjusting a pre-trained model to perform better on a specific task.
Fairness	Ensuring the model does not generate biased or discriminatory content.

9. Glossary

9.1 Key Terms

- **Joke Generation Model**: A machine learning model developed to generate Turkish jokes based on user input. The model is trained on a dataset of jokes and uses natural language processing (NLP) techniques to create humor.
- **Dataset**: A collection of **1500 Turkish jokes** used to train the joke generation model. The dataset includes a variety of jokes to ensure that the model can generate diverse humor.
- **API (Application Programming Interface)**: A set of protocols and tools that allow the system to interact with other software. The API is used for sending user input to the model and receiving generated jokes.
- **Scrum Methodology**: An agile project management framework that divides the development process into **short sprints**, each lasting around two weeks. Tasks are assigned to team members, and progress is reviewed at the end of each sprint.
- **Epoch**: A complete pass through the entire dataset during the training process of the machine learning model. The joke generation model is trained for up to **100 epochs** to ensure convergence.
- **Coherence**: The quality of the generated joke in terms of making sense and maintaining logical flow within the joke. A coherent joke should be easy to understand and follow.
- **Content Moderation**: The process of screening and filtering out harmful, offensive, or inappropriate content. For this project, content moderation is critical to ensure that the jokes produced are ethical and appropriate.
- **Natural Language Processing (NLP)**: A field of artificial intelligence (AI) focused on enabling machines to understand, interpret, and generate human language. NLP techniques are used in this project to generate Turkish jokes.
- **Accuracy**: A measure of how well the joke generation model produces relevant and humorous jokes. In this project, accuracy is defined as the percentage of generated jokes that are considered relevant and high quality.
- **Model Pruning**: A technique used to reduce the size of a neural network by removing weights that have little impact on the model's performance. This is useful to optimize the model for faster processing and lower resource consumption.
- **Fine-tuning**: The process of adjusting a pre-trained model on a new dataset. In this project, the joke generation model will be fine-tuned using the Turkish jokes dataset to improve its performance.
- **Fairness**: The principle of ensuring that the model does not produce biased or discriminatory outputs. This project includes fairness considerations to prevent the model from generating offensive or harmful jokes.
- **User Feedback Loop**: A system that allows users to provide feedback on the quality of the jokes they receive. This feedback is used to improve the model's performance over time.
- **RESTful API**: A type of web service that uses standard HTTP methods (GET, POST, PUT, DELETE) to interact with users or other systems. The joke generation model will be accessed through a RESTful API.