FıkraBot:

A Turkish Joke Generation System with LLM

Student Name	Student ID
Melih Küçük	21 07 06 048
Hüseyin Eray Kızılkaya	22 07 06 302
Nurşeyda Doğan	21 07 06 047
Onur Ulaş Canpolat	20 07 06 314
İdil Öztürk	21 07 06 026
Afnan Mohammed	21 07 06 811

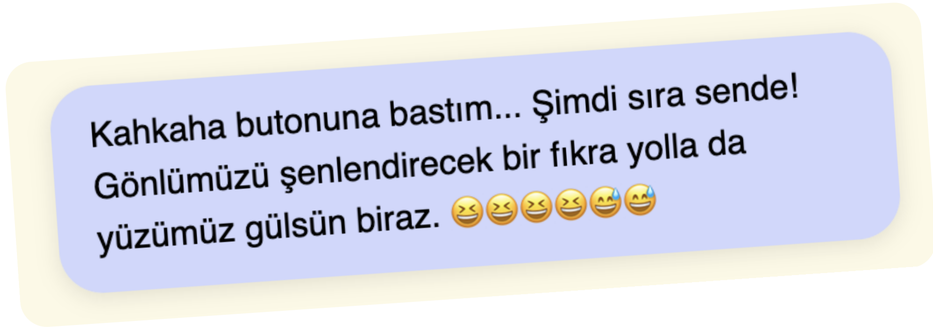
Advisor: Ensar Gül

Introduction and Project Overview

Turkish Joke Generator is a natural language generation project designed to create culturally relevant, coherent, and humorous short jokes in Turkish.

The main objective was to build a fast, reliable, and deployable AI-powered system that generates high-quality jokes with a single button click-without requiring any user input or customization.

The project not only focuses on linguistic accuracy and humor relevance, but also emphasizes output safety, content filtering, and consistent formatting, ensuring that every generated joke is both appropriate and contextually natural for Turkish-speaking audiences.



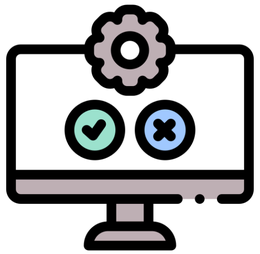
System Features & Functionality

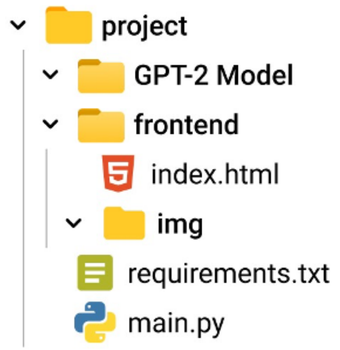
The application features a minimalist, single-button interface that allows users to generate one Turkish joke at a time with zero complexity.

On the backend, the system handles prompt construction, model inference, blacklist filtering, and optional refinement to ensure that each joke is short, contextually appropriate, and free from offensive or irrelevant content.

Since no user input is needed, the system maintains a clean, streamlined user experience and produces consistently structured jokes.

All generation parameters-such as temperature, max tokens, and sampling strategy-are fixed to maintain high output stability and prevent random or incoherent results.











Kahkaha butonuna bastım... Şimdi sıra sende! Gönlümüzü şenlendirecek bir fıkra yolla da yüzümüz gülsün biraz. 😆😆😆😆😅😅

Fıkra Üret







Team Structure and Role Distribution

Our team was composed of six members, each contributing in distinct areas.

Melih served as the team leader, overseeing AI model training and full-stack development.

Hüseyin focused on technology research, data preparation, and contributed to the user interface.

Nurşeyda was responsible for designing and implementing the main frontend.

Onur supported the team with data cleaning and presentation tasks.

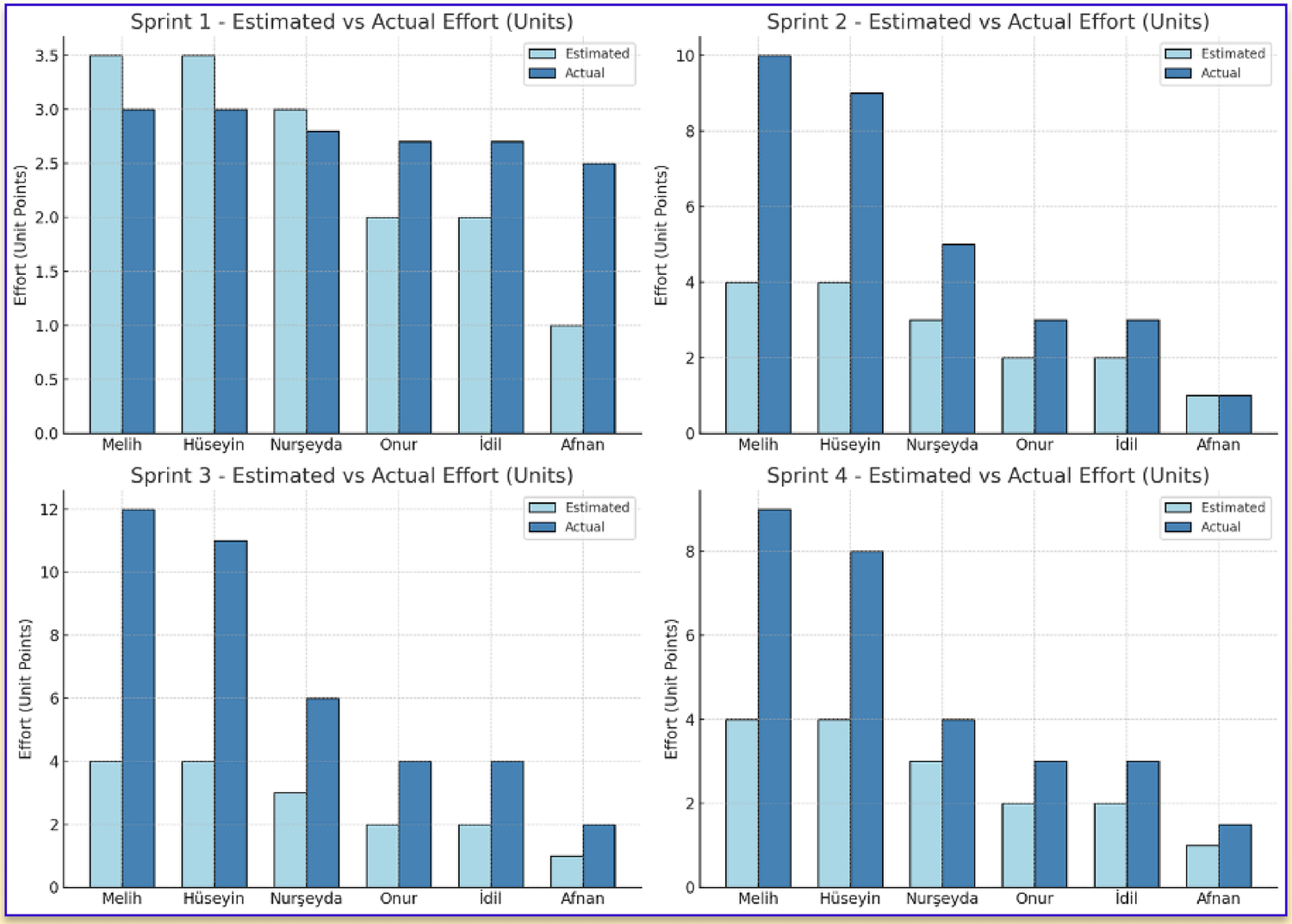
İdil assisted in frontend development and conducted social context research.

Afnan contributed to visual design and helped with dataset collection.

iter no/ developer	Melih (leader)	Hüseyin	Nurşeyda
iter 1	Conducted model research and experimentation	Researched technologies and project-related concepts	Researched social engineering concepts and databases
iter 2	Performed data cleaning, organization, merging, and consolidation Led model training and iterative improvements	Collected and cleaned data	Collected and organized project data
iter 3	Managed the AI-related tasks, including training and evaluation Developed backend functionalities	Participated in training the model Supported backend development	Designed initial wireframes on Figma Implemented core frontend components
iter 4	Improved frontend layout and responsiveness Tuned generation parameters for better outputs Contributed to final reporting and presentation	Assisted in frontend design improvements (layout + responsiveness) Created visual diagrams for documentation and presentation	Helped with final report preparation

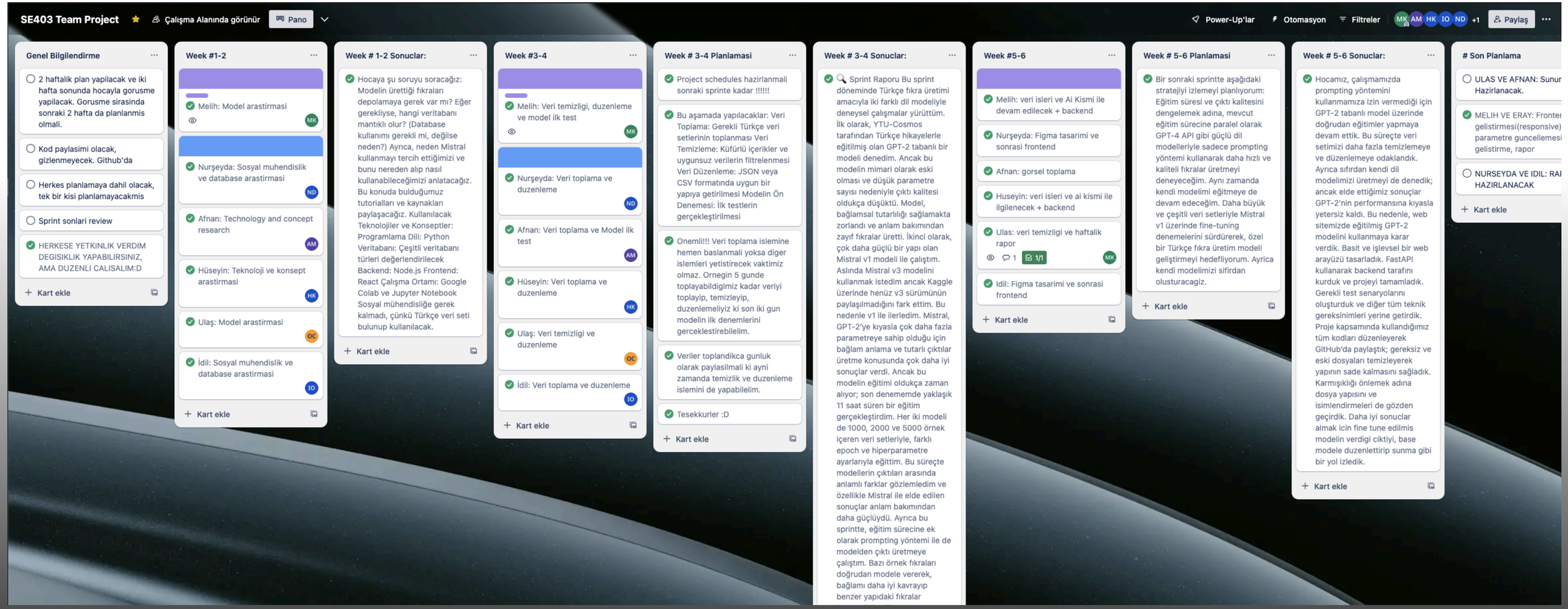
iter no/ developer	Onur	İdil	Afnan
iter 1	Contributed to model research	Researched social engineering topics	Researched relevant technologies and concepts
iter 2	General research	Collected data for the project	Collected data
iter 3	Performed data cleaning tasks	Assisted with core frontend development	Visual materials for the website
iter 4	Assisted with presentation preparation	Feedback for Report	Feedback for Report

Estimated vs Actual Effort



Trello:

https://trello.com/b/xmd1LbLC/se403-team-project

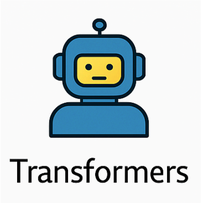


Technologies Used

We used Python as the primary programming language for the backend development. FastAPI was implemented to build the API endpoints for efficient communication between the frontend and the model.

PyTorch and Hugging Face Transformers were utilized for model loading and inference. For fine-tuning, we adopted the PEFT library to apply the LoRA method efficiently.

The frontend was developed using basic web technologies, including HTML, CSS, and JavaScript, with a focus on simplicity and responsiveness.

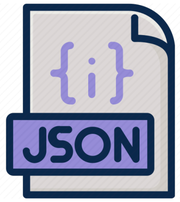


Dataset and Data Cleaning Process

We initially collected approximately 5,000 Turkish jokes from various open-source and semi-structured sources. After a thorough cleaning and preprocessing phase, the dataset was reduced to around 3,600 high-quality and usable samples. During this process, we removed content that included offensive language, explicit references, dates, URLs, or any form of violence.

The final dataset was standardized and formatted consistently in both .csv and .json formats to ensure compatibility with training workflows.

	text
0	Nasreddin hoca traş olmak için berbere gider. ...
1	Nasreddin hoca bir gün yolda giderken bir adam...
2	Nasreddin hoca bir gece aniden uyanır: "hanım ...
3	Nasreddin hoca bir gün öğle uykusundayken dışa...
4	Anadolu selçuklu sultanı, nasreddin hoca'nın ş...
5	Yakın köylerden birinde oturan ve nasreddin ho...

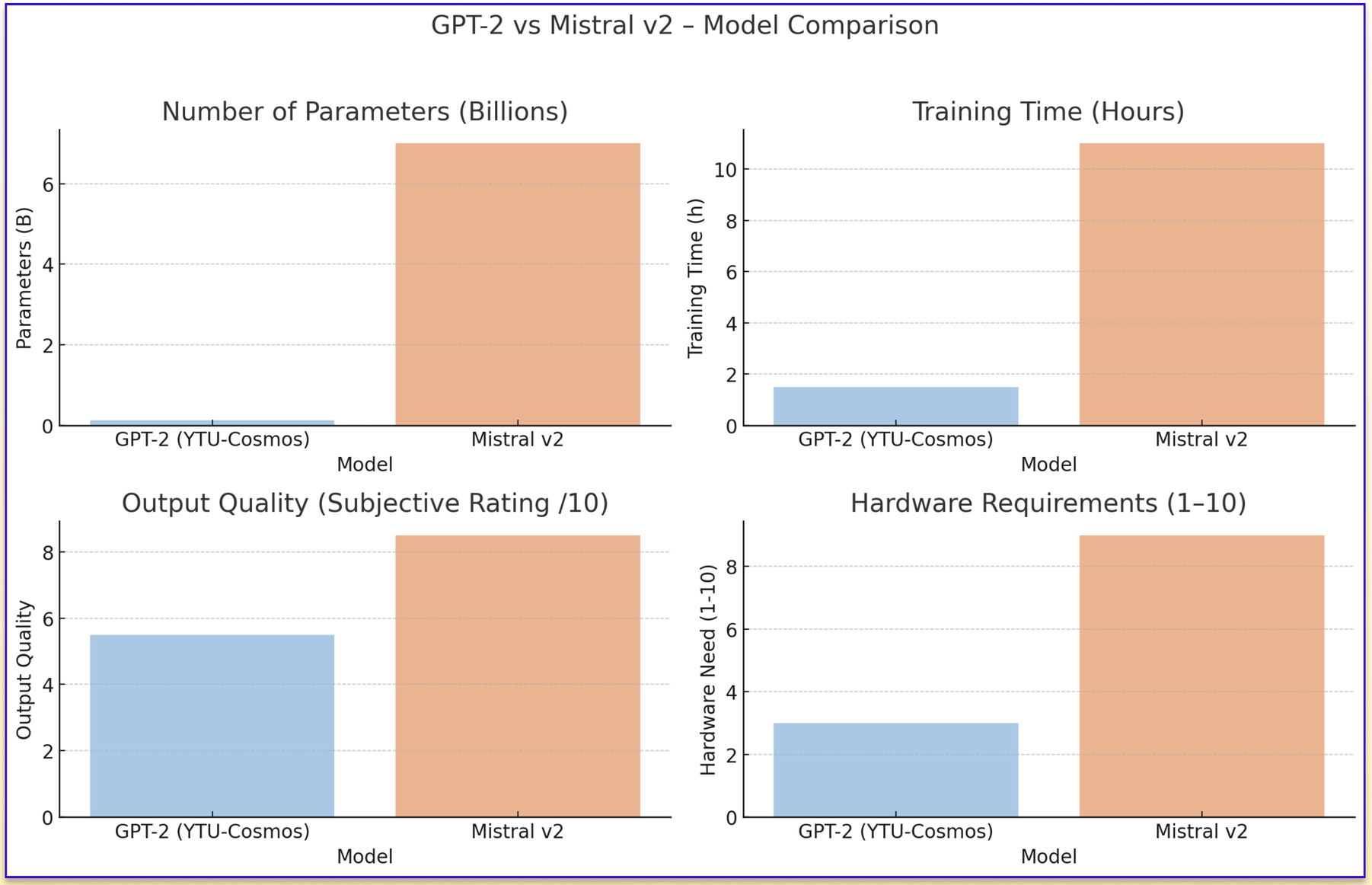


Model Selection and Training

We fine-tuned both the GPT-2 model (YTU Cosmos version) and Mistral v2 using our curated joke dataset.

While GPT-2 was significantly faster in terms of training and inference time, the quality and coherence of its outputs were limited, often lacking contextual fluency. On the other hand, Mistral v2 delivered more context-aware and semantically consistent jokes, but required substantially more computational resources and longer training durations.

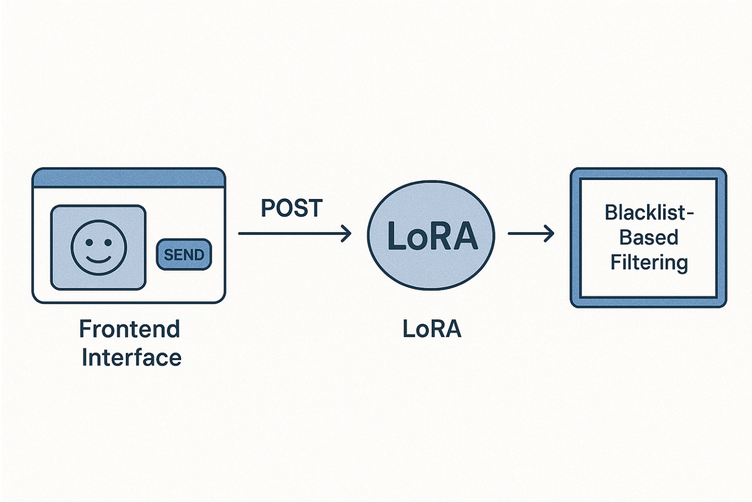
Additionally, we conducted experimental trials by training a language model from scratch, primarily to understand the fundamentals of model architecture and data-to-output transformation.


GPT-2 vs Mistral v2 – Model Comparison
Number of Parameters (Billions)
Parameters (B)
0
2
4
6
GPT-2 (YTU-Cosmos)
Mistral v2
Model
Training Time (Hours)
Training Time (h)
0
2
4
6
8
10
GPT-2 (YTU-Cosmos)
Mistral v2
Model
Output Quality (Subjective Rating /10)
Output Quality
0
2
4
6
8
GPT-2 (YTU-Cosmos)
Mistral v2
Model
Hardware Requirements (1–10)
Hardware Need (1-10)
0
2
4
6
8
GPT-2 (YTU-Cosmos)
Mistral v2
Model

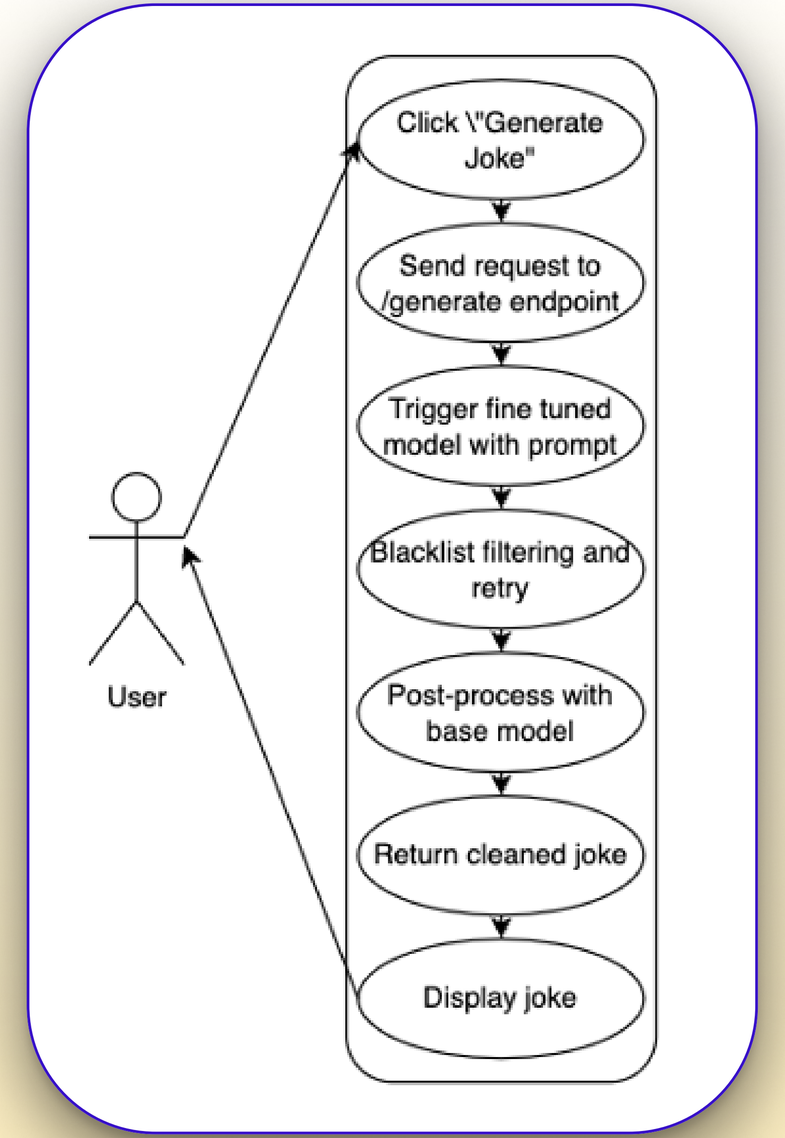
Backend – Frontend Integration Process

The system features a minimalistic frontend interface where a single button click triggers a POST request to the FastAPI backend. Upon receiving the request, the backend applies a predefined prompt and invokes a LoRA fine-tuned language model to generate a joke.

The generated output is then subjected to a blacklist-based filtering mechanism to eliminate inappropriate or irrelevant content. If necessary, the result is further refined using the base version of the model to enhance coherence and fluency. Finally, the polished joke is returned to the frontend and displayed to the user.



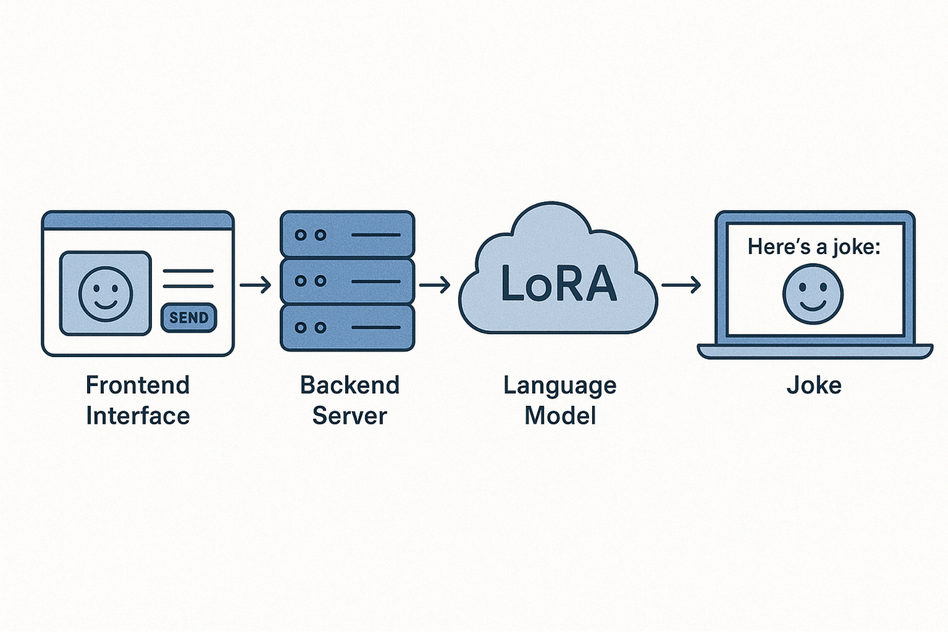
FastAPI

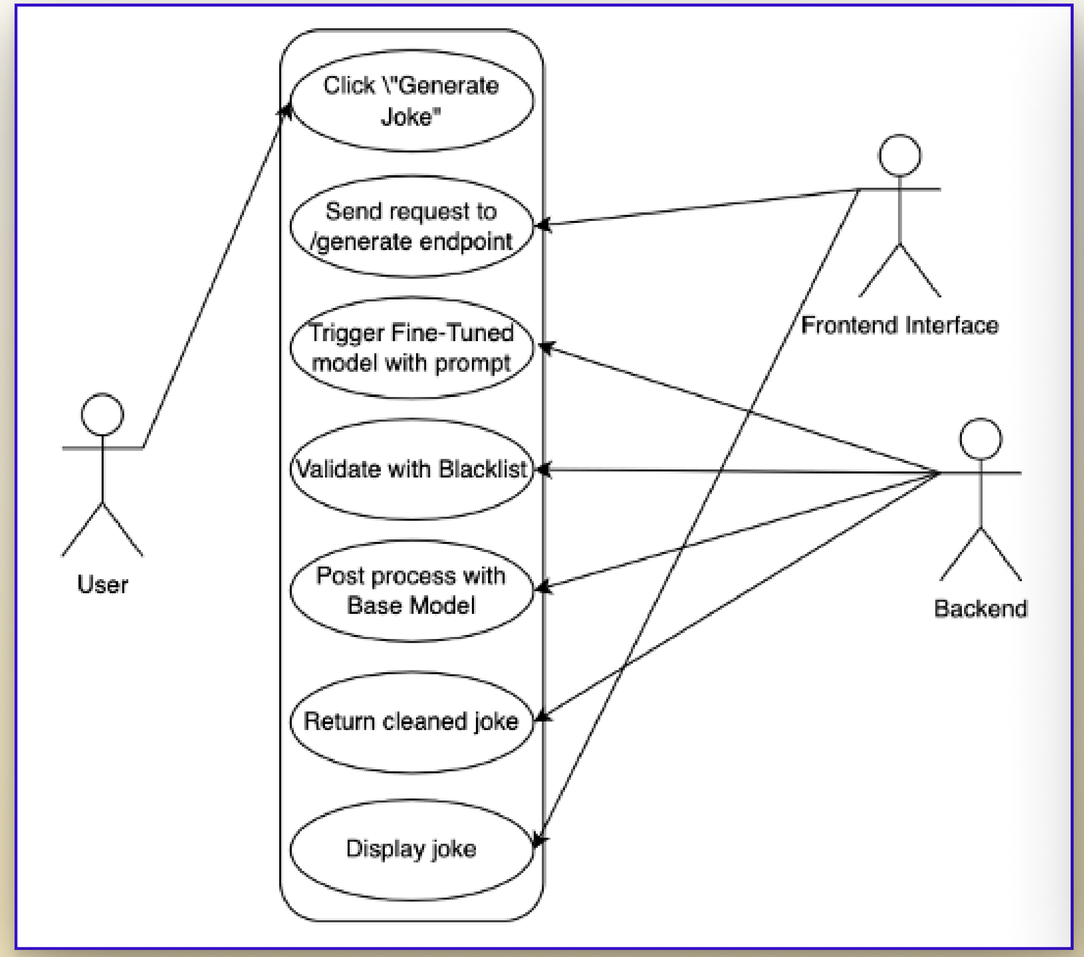

Click \"Generate Joke"
Send request to /generate endpoint
Trigger fine tuned model with prompt
Blacklist filtering and retry
Post-process with base model
Return cleaned joke
Display joke
User

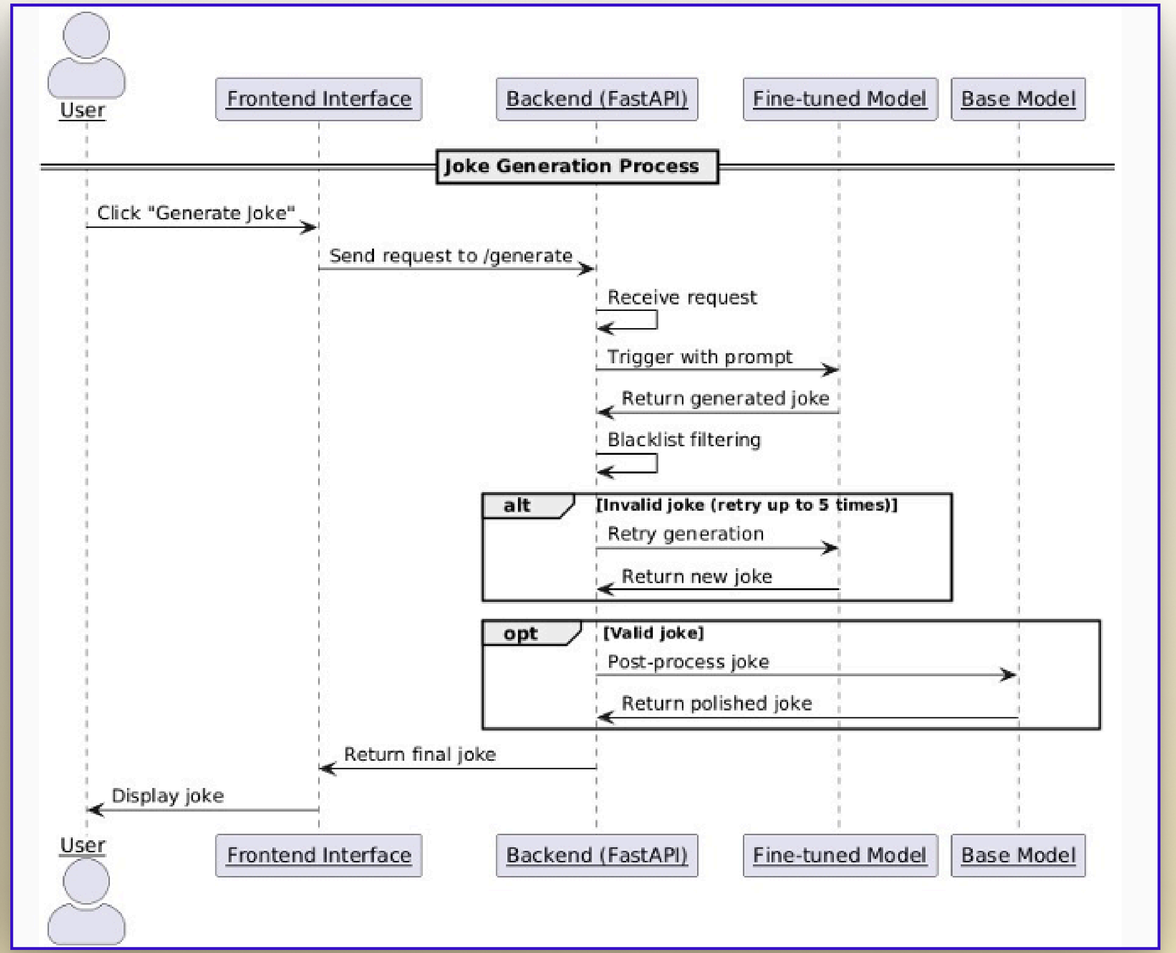
System Flow and Architecture

The system architecture is composed of four main components: a user-friendly frontend interface, a FastAPI-based backend server, a LoRA fine-tuned language model responsible for initial joke generation, and a base model utilized for optional post-editing.

When a generation request is triggered, the backend processes it through these components in sequence-first generating a joke, then filtering it for safety and relevance, optionally enhancing it with the base model, and finally returning the finalized output to the frontend for display.



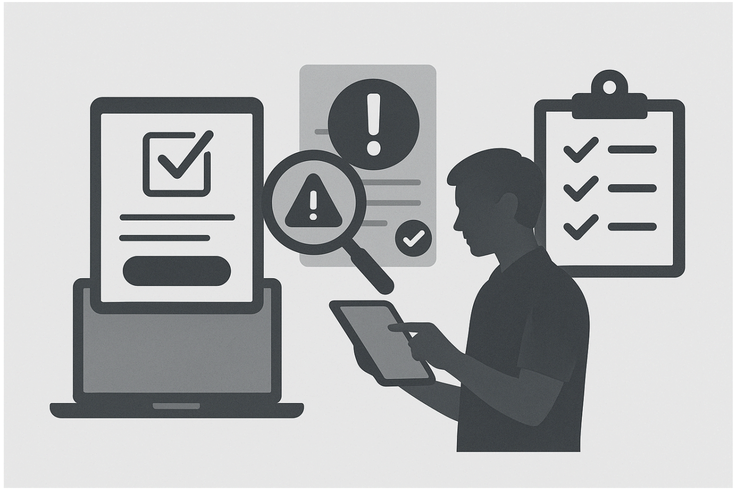

Click \"Generate Joke"
Send request to /generate endpoint
Trigger Fine-Tuned model with prompt
Validate with Blacklist
Post process with Base Model
Return cleaned joke
Display joke
User
Frontend Interface
Backend


User
Frontend Interface
Backend (FastAPI)
Fine-tuned Model
Base Model
Joke Generation Process
Click "Generate Joke"
Send request to /generate
Receive request
Trigger with prompt
Return generated joke
Blacklist filtering
alt
[Invalid joke (retry up to 5 times)]
Retry generation
Return new joke
opt
[Valid joke]
Post-process joke
Return polished joke
Return final joke
Display joke

Testing Process

We manually tested each function using Postman and frontend. Tests included blacklist validation, retry logic, joke length, and method restriction. We conducted manual testing of all key functionalities using both Postman and the web frontend.

The tests covered critical aspects such as blacklist pattern validation, retry mechanism behavior for invalid outputs, enforcement of maximum joke length, and restriction of unsupported HTTP methods. Each function was verified to ensure that the system responds correctly under expected and edge-case scenarios.



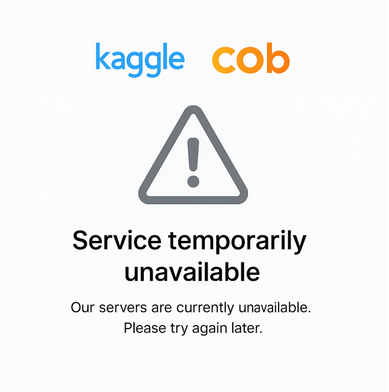
Test ID	Requirement	Precondition	Test Steps	Expected Result	Result
TC01	Should return a meaningful joke via valid POST request	Server running, model loaded	Send POST to /generate with JSON payload	A logical and relevant joke should be returned	✓
TC02	Joke must not include blacklisted content (e.g. links)	Validation function active	Check output for words like http, tweet, etc.	No blacklisted words should appear in output	✓
TC03	Joke must not exceed 30 words	Tokenizer and model configured	Count number of words in the joke	Joke should be within 30 words	✓
TC04	Retry logic should trigger and exit gracefully	Retry limit set, force invalid outputs	Trigger up to 5 failed generations	Should return: "No suitable joke could be generated"	✓
TC05	Edited joke should be shorter and more humorous	Base model editing enabled	Run generation + post-editing step	Final joke should be clearer, shorter, and funnier	✓
TC06	Only POST requests should be allowed	FastAPI app running	Send GET request to /generate	Should return HTTP 405 Method Not Allowed	✓
TC07	Model loading failure should be logged, not crash app	Faulty adapter path used	Launch server with invalid model path	Error should be logged, app must not crash	✗

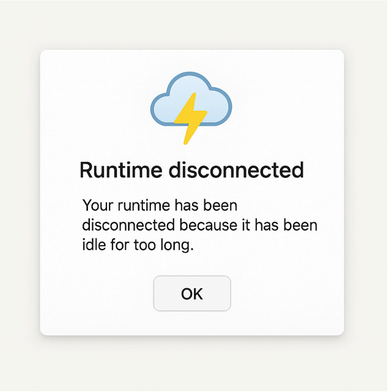
Risks and Challenges

Throughout the project, we encountered several challenges, including limited computational resources, extended training durations, and difficulties in maintaining consistent data quality.

Selecting the most suitable model architecture was also a non-trivial task, given the trade-offs between performance and hardware constraints.

Additionally, the use of the GPT-4 API was not approved due to institutional limitations, and the availability of large-scale, high-quality Turkish-language datasets was quite restricted, which further complicated the fine-tuning process.





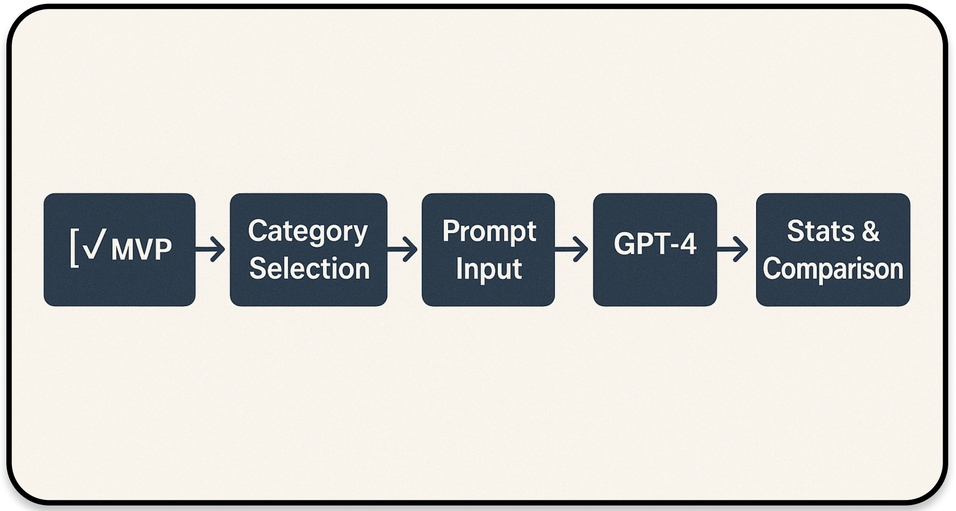
Risks Table

Risk ID	Risk Description	Likelihood	Impact	Mitigation Strategy
R1	GPT-4 API usage was not approved by the instructor	Medium	High	Switched to open-source models like GPT-2 and Mistral
R2	Hugging Face API failure due to authentication issues	Medium	Medium	Used huggingface-cli login; tested models locally before deployment
R3	Compatibility issues with MPS backend on macOS	Low	Medium	Optimized inference with torch.device("mps") and float32 precision
R4	Dataset too limited for reliable user-prompt generation	High	High	Switched to fixed internal prompting with random output generation
R5	Inappropriate or unsafe joke outputs	Medium	High	Implemented regex-based blacklist and post-editing with base model
R6	Training interruptions and instability on Google Colab	High	Medium	Migrated training process to Kaggle for longer, more stable sessions
R7	Initial plan to use a database was deemed unnecessary	Low	Low	Confirmed with instructor and excluded database functionality

Achievements and Future Plans

Through this project, we gained hands-on experience in full-stack development, language model training, dataset curation and cleaning, and collaborative project management using Agile principles. Each team member had the opportunity to engage with real-world challenges in AI development.

Moving forward, we plan to enhance the system by enabling category-based joke generation, introducing user-defined prompting features, and integrating more advanced models such as GPT-4 for improved output quality and versatility.



Live Demonstration

References

1. **Hugging Face**. “Transformers Documentation.”https://huggingface.co/docs/transformers
2. **Hugging Face Hub** – Model Page. ytu-ce-cosmos/turkish-gpt2-large.https://huggingface.co/ytu-ce-cosmos/turkish-gpt2-large
3. **Google Colab**. “Colaboratory Documentation.”https://colab.research.google.com
4. **Kaggle**. “Kaggle Notebooks and Dataset Platform.”https://www.kaggle.com
5. **GitHub**. “FastAPI Official Repository.”https://github.com/tiangolo/fastapi
6. **PEFT**: Parameter-Efficient Fine-Tuning Library.https://github.com/huggingface/peft
7. **PyTorch**. “Torch Documentation.”https://pytorch.org/docs/stable/index.html
8. **ChatGPT (OpenAI)**. Used interactively for debugging assistance, language modeling support, and report drafting suggestions.
9. **Stack Overflow Community**. Various posts regarding FastAPI, JSON handling, training issues, and PyTorch deployment.
10. **MDN Web Docs**. “Frontend responsiveness and layout strategies.”https://developer.mozilla.org/

https://github.com/Maltepe-University-SWEng/term-project-team-8

Thank you for listening.

Happy to answer your questions.